Multiprocessor control and data analysis for a coherent auroral radar

Foster, David George

MULTIPROCESSOR CONTROL AND DATA ANALYSIS FOR A COHERENT AURORAL RADAR

by

David George Foster, B.Sc.

A thesis submitted in accordance with the regulation for the degree of Doctor of Philosophy in the University of Durham Department of Applied Physics & Electronics

1982

Summary Of Thesis

The results from the STARE (Scandinavian Twin Auroral Radar Experiment) have been able to indicate how the irregularity drift velocities vary, over a large viewing area, in the auroral zone ionosphere. Many inferences have been made concerning localised irregularity structures, particulaly the high velocities (1 meter irregularities having a velocity much greater than the ion acoustic velocity) that have been observed. To further elucidate the many questions that have arisen concerning longitudinal variations of the irregularity structure, a second, similar radar was designed to observe an area South West of the STARE viewing area. This new radar is called SABRE (Swedish And British Radar Experiment). The siting of the radar viewing areas are such, that at times the STARE radar is observing inside the auroral oval and the SABRE radar is observing outside the auroral oval. This will provide interesting information on the irregularity structure across the longitudinal boundary of auroral activity. SABRE was designed not only to help in the ionospheric physics field, but also to provide a technological insight into the design of remote experiments. Many significant advances have been incorporated to overcome the problems that the simple, yet relatively unreliable, STARE computer system had. In particular, the SABRE radar has the capability to monitor, through the use of concurrent processing, many system parameters and detect faults as they occur. More importantly a synchronous 2400 baud modem facility is available, with automatic dial up, and this enables not only faults to be reported immediately, but also provides the

capability to control the experiment remotely. By this method data may be transferred, in real time, to the analysis site for processing. By the use of a multi-computer system with a hardware inter-processor communications interface this may be achieved without impeding the data processing throughput at all. To maintain the reliability of the system, power failures are catered for by re-loading the system software automatically from bubble memory. This device has proved extremely reliable in operation, considerably more than a disk unit. The bubble memory is also used to keep a record of the voltages and currents monitored. During a failure the computer system may initiate a predetermined sequence to shut down the radar, or modify some of the parameters (i.e. EHT voltage may be monitored and controlled) to correct the condition. It will then 'dial up' the analysis site and dump the data in the bubble memory for examination. In this way it is possible to gain some insight into the cause of the failure. In case of catastrophic failure the bubble module may be unplugged and taken away for analysis.

CONTENTS

Chapter 1

Introduction and Background

1.0 Introduction

The history of atmospheric phenomena and meteorological optics goes back thousands of years, and references to events such as rainbows and halos appear in every ancient literature. By Newton's time, the gross features of many visible phenomena had been explained. The rainbow alone remained a mystery until 1803 when Thomas Young used the supernumary rainbow effect to support his theory of interference. Investigations in the nineteenth century began to elucidate the detail of the atmosphere, and in particular the existence of "regions" or "layers" with different properties.

1.1 The Ionosphere

The existence of one such layer, the ionosphere, as an electrically conducting region was first proposed by Balfour Stewart in 1883. He inferred it from a study of the small daily geomagnetic variation observed at the earth's surface. In the opening years of this century, shortly after Marconi had transmitted radio waves across the Atlantic, the presence of an ionised region in the upper atmosphere was jointly proposed by Kennelly and Heaviside. Appleton recognised the presence of more than one ionised layer and to him are due the names, E and F (and the parts F1 and F2). Later the D region was discovered to extend below the E region.

The formation of the ionosphere is largely due to the action of the emitted radiation from the sun on the atmosphere. Short wave ultra violet radiation extending into the X-ray part of the spectrum interacts with the molecules and atoms, primarily those of nitrogen and oxygen, raising them to a higher energy state by removing an outer electron leaving the positive ion. The ionosphere is a region where there is a significant number of these free electrons and extends down to as low as 60km from the earth's surface. The electron density is greatest above 100km and reaches a maximum at the peak of the F2 layer.

1.2 The Aurora

One of the most spectacular manifestations of the interaction of extra terrestrial particles with the atmosphere, is known as the aurora. The flourescent luminosity of the aurora is the result of atoms, excited to a higher energy level, relaxing to a lower and more stable energy level, consequently re-radiating the energy as electromagnetic radiation in the visible part of the spectrum. The polar situation of the aurora is a consequence of the deflecting influence of the earth's geomagnetic field. High energy particles, particulaly dissociated hydrogen, on encountering the magnetic field, spiral along the field lines towards the polar regions. The details of this complicated process have been described in many texts on the subject and will not be discussed further here. The effect of the aurora on radio propagation was studied in 1933 during an expedition to Tromso, Norway by Appleton whose observation of the ability for the highly ionised regions of the ionosphere to

scatter radio waves started the study of the auroral zones magnetic and electric field properties.

1.3 Radar Observations

Two types of ionospheric radar are currently being used extensively. The first type, commonly called incoherent radar, ususally have geographically separate transmitting and receiving sites. These radar require large transmitter powers, of the order of 1000kw, since the returned power after scattering by the medium is so small. These type of radar will not be considered in any great detail here as they have been described in very great detail in the literature (Evans, 1974; Carpenter, 1975).

The second type are termed coherent backscatter radar. These type of radar have the transmitting antenna and receiving antenna co-located at the same site. The radar is termed backscatter radar as the radio waves of interest are those scattered back towards the transmitting antenna. The term coherent comes from the fact that, unlike incoherent radar, the received signal is that which is unmodified in frequency from the transmitting signal and bears a phase relationship with the transmitted signal. One of the first phenomena noted by workers using this type of auroral radar, as it has now become known, was the aspect angle sensitivity of the returned signal, that is to say, the strength of the radar returns as a function of the angle between the radar wave vector and the magnetic field lines. In particular, the strength of the returns tended to decrease as the angle in question was varied from orthogonality. The first experimental evidence of this effect came from the Geophysical

Institute of Alaska with the aid of three radar operating at 25, 50 and 100 MHz. Chapman (1953) proposed that the sensitivity could be explained by considering the scattering medium as columns of ionisation parallel to the earth's magnetic field giving maximum echo when viewed orthogonal to the earth's field. The considerations of his work suggested a very sensitive dependence and indeed predicted that some sites would be totally unsuitable for receiving backscatter. Notably two such sites that fell into this catagory were College (Alaska) and Tromso (Norway). In actual fact these two sites are very good for receiving radar aurora. Booker (1955) attempted to modify the existing theory by considering columns of ionisation of restricted length in both longitudinal and transverse directions. He proposed that the creation of these columns was attributed to atmospheric turbulence in the E region and above. Leadebrand et al (1965) investigated the aspect angle sensitivity in great detail with a VHF and UHF radar located in Scotland. The wavelength dependence of radar backscatter was investigated by Presnell et al (1959) and by Leadebrand et al (1965) in detail showing that the efficiency of the aurora to backscatter signals falls off sharply as the frequency of the radio waves increase. An exact dependence was proposed by Flood (1960) after studying data from three radar sited at Buffalo, New York. This work was followed up by Leadebrand (1967) who showed that the reflection coefficient varies as a power law <<<<<<<<<<< in the 400 to 3000 MHz range, thus favouring study of the aurora at lower frequencies. It became clear now that the choice of frequency of the radar is closely related to the nature of the irregularities

to which it will be sensitive. Balsley (1971) indicated that the radar is most sensitive to those irregularities whose spatial Fourier component is half the radar wavelength. Thus a 50 MHz radar would be sensitive to 3 metre irregularities.

Early spectral observations of radar aurora by Bowles (1960) and later theoretical work by Farley (1963) indicated that a two stream system of plasma instability could account for the characteristics of the observed irregularities. This new theory showed that the irregularities should arise spontaneously in regions where a sufficiently strong current is flowing normal to the magnetic field. These irregularities would have a strong alignment with the magnetic field and a wide range of wavelengths possible. Later experimental studies by Balsley (1969) showed that there were two types of irregularities that could cause radar aurora. The first type, known as type 1 instabilities, are the two stream irregularity described earlier and were known to travel at the ion acoustic velocity, about 360 m/sec. The work by Balsley (1969) showed radar returns with a Doppler frequency shift considerably smaller than that predicted for the type 1 instability. Balsley (1969) attributed this new type of scattering instability, now called type 2 irregularities, to some type of ionisation gradient and he showed that the velocity of propagation was the same as the electron flow velocity in the region where the instability occurred. Balsley (1969) also noted that the irregularity flow was closely related to the current system known as the auroral electrojet and he showed the existence of the nightime electrojet by noting the West to East

motion of the type 2 irregularities. This work was followed by Greenwald et al (1975) who used a 50 MHz backscatter radar to monitor the motion of radar aurora simultaneously with electrojet current flow deduced from a set of four ground based magnetometers in Alaska. This work also indicated a correlation between the intensity of the scattered signal, as measured by the radar, and the magnitude of the electrojet current. The advantages of being able to monitor electrojet current systems directly using auroral radar was reviewed in detail by Greenwald (1979) but clearly the ease of interpretation of radar aurora is a great improvement over magnetometer techniques that have to contend with many secondary factors, such as induced earth current effects. Ecklund et al (1975) described a radar that was one of the first successful attempts to produce information about a scattering volume of aurora by simultaneously observing backscatter from two different directions. This new technique provided information about the spatial structure of the irregularities. Greenwald (1978a) extended this basic "crossed beam common volume bistatic" radar idea with an experiment sited in Northern Scandinavia. STARE (Scandinavian Twin Auroral Radar Experiment), as it is known, employed a phased array receiving antenna which essentially gave eight separate receiving beams. The use of such a system at the two sites involved gave a common volume composed of 64 crossing points of the beams. A full description of the STARE system and the data analysis involved will be given in the next chapter.

It was known at this time that the unstable waves constituting the irregularities propagated in a plane perpendicular to the earth's magnetic field and that this applied to both type 1 and type 2 irregularities. Rogister and D'Angelo (1970) combined the equations describing type 1 and type 2 irregularities into a single dispersion relation that described the physical situation more closely than either type 1 or type 2 equations did singly. Greenwald (1978a) was able to show that, from initial observations with the STARE system, the irregularities propagated with the same velocity as the electron drift. One assumption that was made is that the observed Doppler velocity was related to the true irregularity drift velocity by the cosine of the angle between the true irregularity drift direction and the radar wave vector. Ecklund (1977) gave evidence for the assumption that the irregularity drift velocity was indeed the same as the electron drift velocity, which in the E region is given by ExB/B^2 where E and B refer to the electric and magnetic field intensities respectively. This has lead Greenwald (1978a) to the conclusion that the electric field vectors may be calculated in a very simple manner from the irregularity drift velocity. Satellite experiments, that are able to measure the point electric field directly, simultaneous with STARE observations (Greenwald, 1978b) have reaffirmed that the electric field may be measured simply with an auroral radar.

Following the success of STARE in being able to provide information on the electric field structure of the ionosphere over a very large (300,000 km^2) common volume a second, similar system was proposed. The new system, to be known as SABRE

(Swedish And British Radar Experiment), was to complement the existing STARE system. The location of the STARE radar are in Malvik, Norway and Hankalsalmi, Finland with a viewing area over Northern Scandinavia. The location of the SABRE radar are in Uppsala, Sweden and Wick, Scotland with a viewing area South West of the STARE radar, as shown in figure 1.1. The criterion of perpendicularity of the radar wave vector and the magnetic field chiefly determined the location of the sites. The Uppsala radar is very conveniently sited at an ionospheric research institute, but the Wick radar is sited in a very beautiful, deserted and inaccessible corner of the Scottish highlands. This has proved an important consideration when designing the radar and many of the features included in the computer system are as a consequence of this remoteness. The viewing area of the SABRE system was chosen to provide information on the longitudinal variation of the irregularity structures, as can be observed from the results from the STARE and SABRE radar that have been taken concurrently. Other benefits from having two such systems operating simultaneously have been outlined by Greenwald (1979) and include, observations of PC5 magnetic pulsations, field aligned currents, and magnetospheric convection. Also the yearly variation of the statistical quantity known as the auroral oval is interesting. The oval, which essentially defines the longitudinal boundary of auroral activity, would at times be inside the STARE viewing area but outside the SABRE viewing area. The results from the four radar would provide very interesting information on irregularity and electric field structures at the edge of auroral activity.

The contributions of the SABRE system in conjunction with the STARE system, and perhaps more such systems, are exciting. They provide a quasi continuous picture of electric field structure in the auroral ionosphere which is not only important for collaberative rocket experiments, but also for the understanding of the upper atmosphere. This thesis largely deals with the details of the SABRE computer system and how it achieves the information processing necessary to provide the physicist with real time, and not merely recorded, data from the radar.

Chapter 2

The STARE Radar System

Quod Semper Quod Ubique
Quod Ab Omnibus Creditum Est

St. Vincent Of Lerins.

2.0 Introduction

This chapter deals with the details of the STARE system. Consideration is given to all aspects of the hardware used so that the differences between the STARE and SABRE system will become clear. The details of the antenna system are described as these are the same for both the STARE and the SABRE radar.

2.1 STARE Hardware

The hardware for each of the STARE radar may be divided into four subsystems for the purposes of discussion. These are the transmitter, antennas, receiver, and computer systems.

The first of these, the transmitter, is a modified television transmitter that is operated at 140 MHz at the Norwegian site and 143.8 MHz at the Finnish site. The transmitter is capable of providing 50 Kw peak pulsed power. The duty cycle, that is the percentage of time that the transmitter is delivering power, is kept below 2 percent. This ensures an output valve life of the order of six months. More details of the transmitter have been descibed in the literature (Greenwald, 1978a).

The aerial system must at this stage be discussed in some detail, since it is this that contributes largely to the originality of the STARE radar. The transmitting antenna system is composed of two vertical stacks of four eight-element Yagi antennas fed in phase from an eight-to-one power divider. The receiving antenna system consists of 64 eight-element Yagi

antennas arranged in 16 vertical stacks. The outputs from the 16 receiving stacks are first amplified, using a very low noise preamplifier, and then fed into a Butler matrix (Delaney, 1961). This device is a passive phasing network that produces at its output the signals that represent the individual beams. The method of achieving multiple beams may be described with reference to figure 2.1. This shows the output from a 3dB directional coupler and from a hybrid ring circulator. When the input voltages have the amplitudes and relative phase angles shown, all the input signal power will appear at the indicated terminal. The simplest multibeam array is formed by using two antenna elements and one hybrid ring or one 3dB coupler. Figure 2.2 shows such an array using a 3dB coupler. An incident wavefront as shown, and only from the two directions indicated, will produce element currents that are 90^{o} out of phase. When this occurrs all the received signal will come out of one of the coupler terminals. A 4 beam matrix can be constructed by interlacing two 2 beam matrices. A typical 4 beam matrix is shown in figure 2.3 and the extra passive phase shifters that are required should be noted. Four outputs are possible, although only two directions of incident beam are shown, and the phases at each point are indicated. The 16 beam matrix is formed simply by cross coupling four 4 element matrices thus forming 16 receiving lobes. In the STARE system only the eight central beams are used. This is because the outermost beams at each end of the receiving array have a sufficiently large sensitivity in an unwanted direction (first order lobes) to make their use unreliable. The central eight principal lobes have side lobes

that are between 15 and 20 dB's down on the principal lobes. This method of beam forming has the advantage over an electronically switched phased array in that it is entirely passive and therefore reliable. This method of beam forming is used in the SABRE system and photographs of the antennas are given in figure 2.4.

The computer system for each of the two STARE sites comprises of a Data General Nova 2 computer, 8K (8x1024) of core memory and several special purpose interfaces. These interfaces are wire wrapped on special card cages that fit into the main chassis of the computer.

The data acquisition interface is the most complicated of the interfaces and is responsible for the collection of data from the receiver and the control of the DMA (Direct Memory Access) channel of the computer. The quadrature outputs from the receiver, for a particular channel, are sampled simultaneously by the high speed A-D (Analogue to Digital) converters used in this interface. The two eight bit values obtained from the A-D converters are combined into a 16 bit word which is then entered into the computer memory by DMA. The next channel is then converted, and the process repeated, until all eight channels have been sampled. The time is kept by a digital clock and an interface is provided to read the clock so that the time may be recorded as the data are taken. A pulse interface is provided that sends a logical 1 (+5V) to the transmitter when power it to be transmitted, and a logical 0 (0V) when it is not. In this way the computer is able to control the pulse length. Data are collected and averaged over a period of time that is usually 20 to 60 seconds in length and this represents the temporal resolution of the system. After this integration period the results are written onto magnetic tape.

2.2 STARE Software

The data are processed by the computer in a manner described by Greenwald and Ecklund (1975) and by Greenwald (1978a). The transmitter is sent a single-double pulse sequence by the computer as shown in figure 2.5. Data are sampled by the A-D interface after a predetermined time period once the pulse or pulses have been sent. This period represents the 'round trip' delay of the pulse sequence from the transmitter to the scattering medium and back. The delay therefore determines the range from which data are taken. The transmitted pulse width (T) is 100 microseconds typically, and this determines the range resolution of the system. When data are taken from the receiver the main requirement is that the sampling of the eight channels should be concluded within the time period T. The receiver is not sampled again until the start of the next time period T. This sampling process is then repeated N times, where N represents the number of ranges from which data are taken. The quadrature outputs for each individual channel are sampled simultaneously so that no extra phase delay is introduced by the sampling process.

For a particular range j and lobe i, the backscattered power, P(i,j) is calculated from the real, R(i,j) and imaginary, I(i,j) components of the received signal by:-

$$P(i,j) = R(i,j)^2 + I(i,j)^2$$

To calculate the doppler velocities of the scattering irregularities the double pulse technique is used. Basically the phase delay of the received signal, that has occurred over the time of the double pulse gap, is measured. This gap is always a multiple of the time T and is usually 300 microseconds or 3T. This is done so that data may be taken every time interval T but may then be autocorrelated by using data that were separated by 3T when the receiver was sampled. As figure 2.5 shows, N+3 samples are taken (for a gap of 3T) so that after autocorrelation the number of samples produced is N, for each of the N ranges required.

The autocorrelation may be explained by considering the output from the receiver as a continuous signal described by the equation e^{iO} with O representing an arbitrary phase angle.

Consider the Pth sample taken to be e^{iO} = S1

and the P+3 sample to be given by e^{iO} = S2

This would be the case for a double pulse gap of 300 microseconds.

Then the complex conjugate of S2 would be e^{-iO} = S2

The complex autocorrelation function is defined to be

$$S1.S2 = e^{iO}.e^{iO} = e^{i(O-O)}$$

Where O is the datum for range j and O is the datum for range j+D and D is the number of time periods T in the gap, in this case D=3. The above autocorrelation product is calculated for every lobe and range and may be represented as:-

$$Re(i,j)+kIm(i,j)=(R(i,j)+kI(i,j)).(R(i,j+D)-kI(i,j+D))$$

k is the square root of minus 1 in this context.

Expanding the above formula to isolate the real, Re(i,j) and imaginary, Im(i,j) results:-

$$Re(i,j)=R(i,j).R(i,j+D)+I(i,j).I(i,j+D)$$

$$Im(i,j)=R(i,j+D).I(i,j)-R(i,j).I(i,j+D)$$

At the end of the integration period, the autocorrelated data are averaged and the mean phase shift for each lobe and range is calculated by:-

$$Ph(i,j)=Tan^{-1}(Im(i,j)/Re(i,j))$$

This quantity is directly related to the doppler velocities of the irregularities for each lobe and range. The output from the receiver is the phase difference between the local oscillator of the receiver and the received signal. In this way the receiver is phase coherent with the transmitted carrier. For a frequency of 140 MHz the wavelength of the carrier is 2.14 meters. What is measured by the autocorrelation process is the phase change over a 300 microsecond period. This corresponds to the same phase change of the transmitted carrier caused by the motion of the irregularities during the 300 microsecond period. For a phase change of 180^{0}, the irregularity must have moved 2.14/4=0.53 of a meter. The wavelength change alone represents a distance of 2.14/2=1.07 meters but the extra factor of two is included because the wave has to travel the extra path length twice, once to the irregularity and once back again. This effectively halves the distance that the irregularity has actually moved. In a period of 300 microseconds this corresponds to a velocity of $0.53/300 \times 10^{-6}$=1800 m/sec. This is the maximum velocity that may be measured without aliasing problems. A wavelength change of greater than 1.07 meters is equivalent to a decrease in wavelength by a lesser amount due to the irregularity moving towards the radar instead of away. These velocity calculations are performed when the data from the radar has been processed at the analysis station.

2.3 Conclusion

This chapter has described the STARE radar system as it exists at the present time. The following chapters will concentrate on the problems that have been encountered during the operation of the STARE radar and how these have been overcome in the SABRE system. Finally suggestions will be made on how the STARE radar computer systems may be improved to incorporate some of the features of the SABRE system.

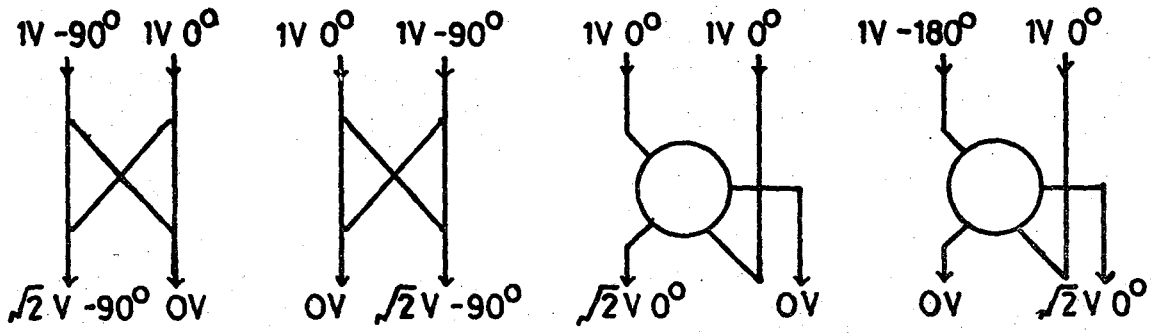


FIGURE 2.1

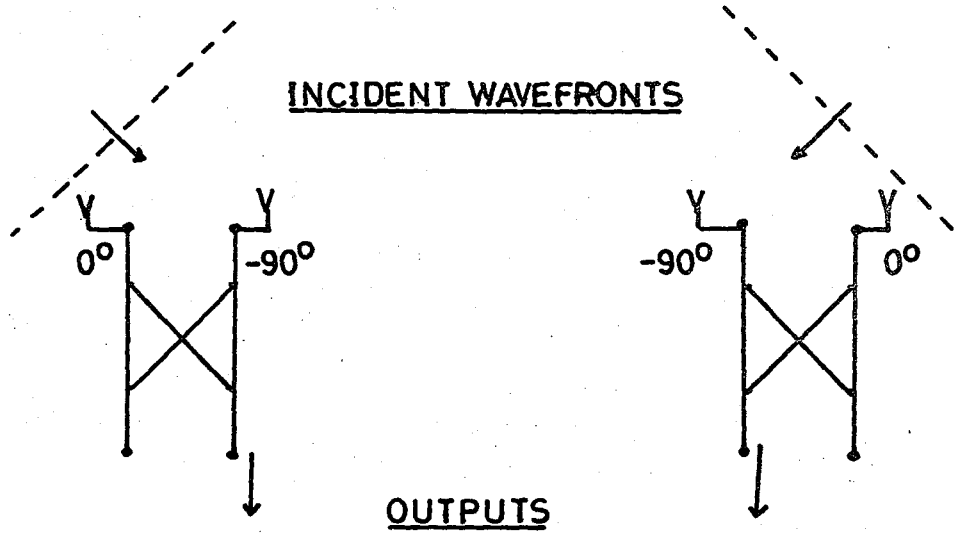


FIGURE 2.2

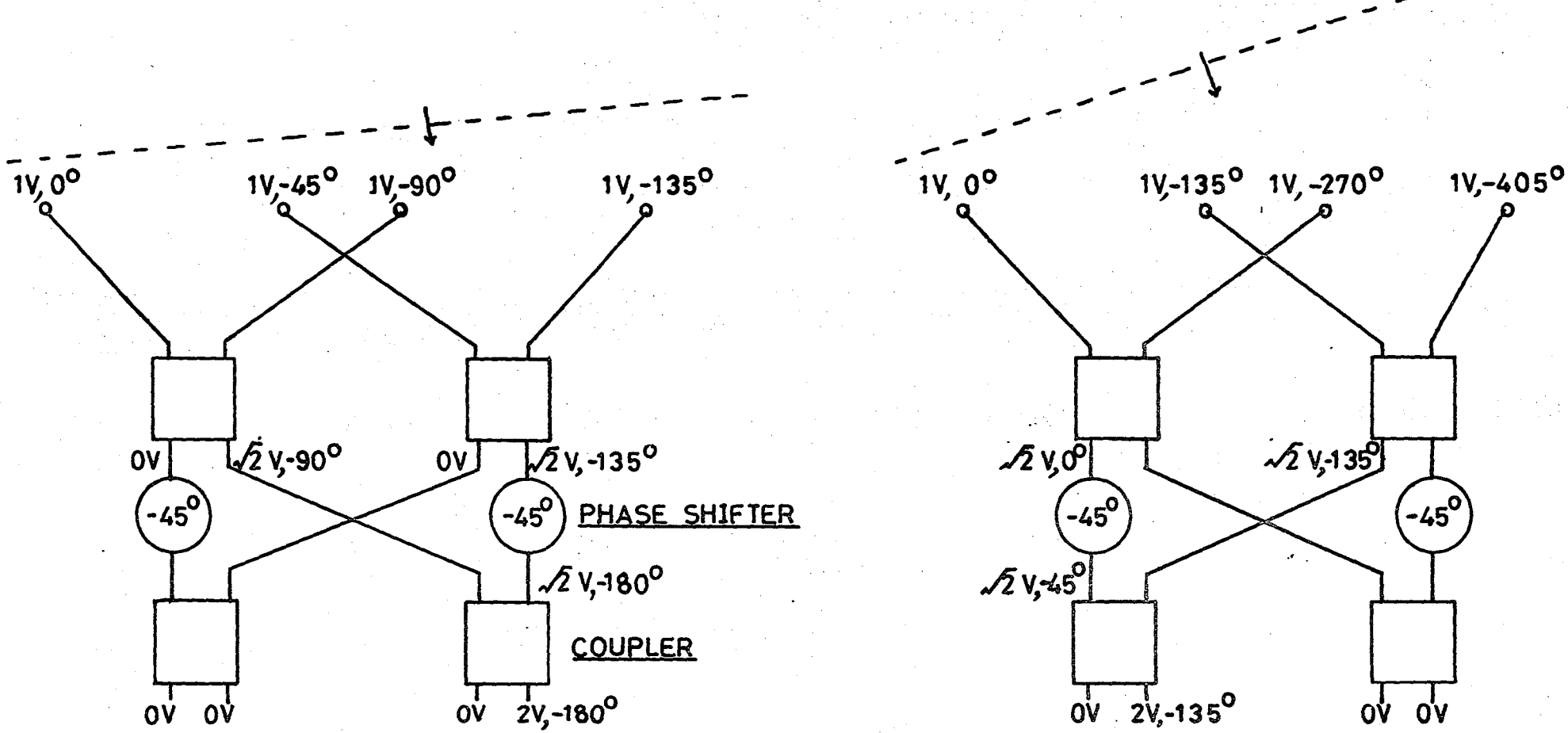
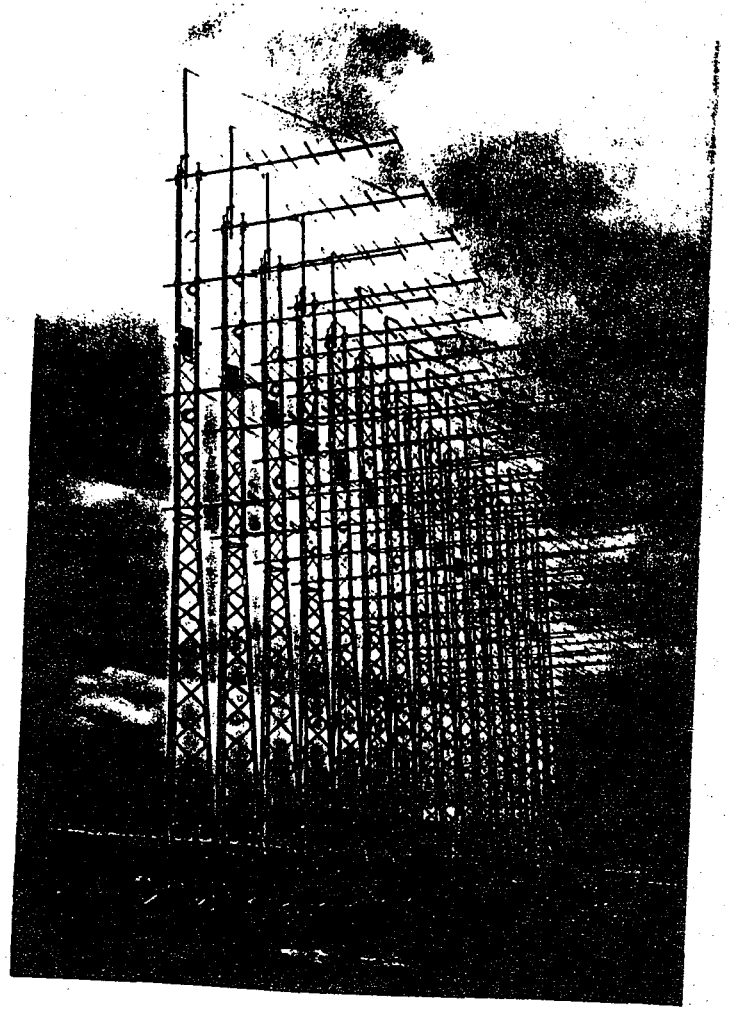
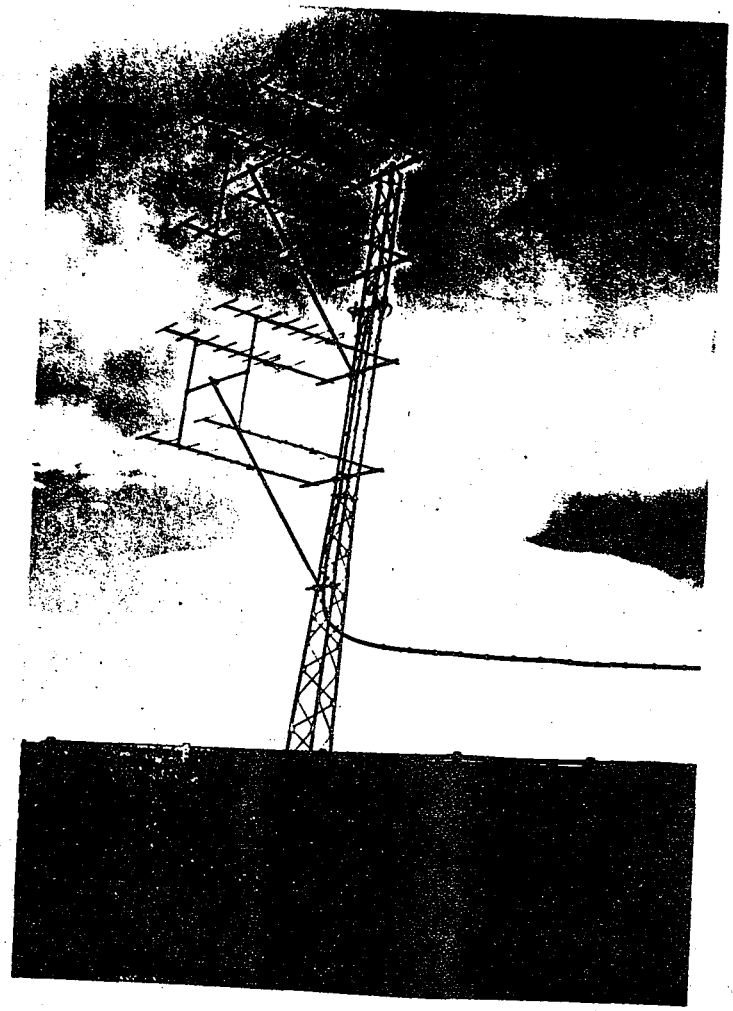

1V, 0°
1V,-45°
1V,-90°
1V,-135°
0V
√2 V,-90°
0V
√2 V,-135°
-45°
-45°
PHASE SHIFTER
√2 V,-180°
COUPLER
0V 0V
0V 2V,-180°
1V, 0°
1V,-135°
1V,-270°
1V,-405°
√2 V,0°
√2 V,-135°
-45°
-45°
√2 V,-45°
0V 2V,-135°
0V 0V

FIGURE 2.3



RECEIVER



TRANSMITTER

FIGURE 24

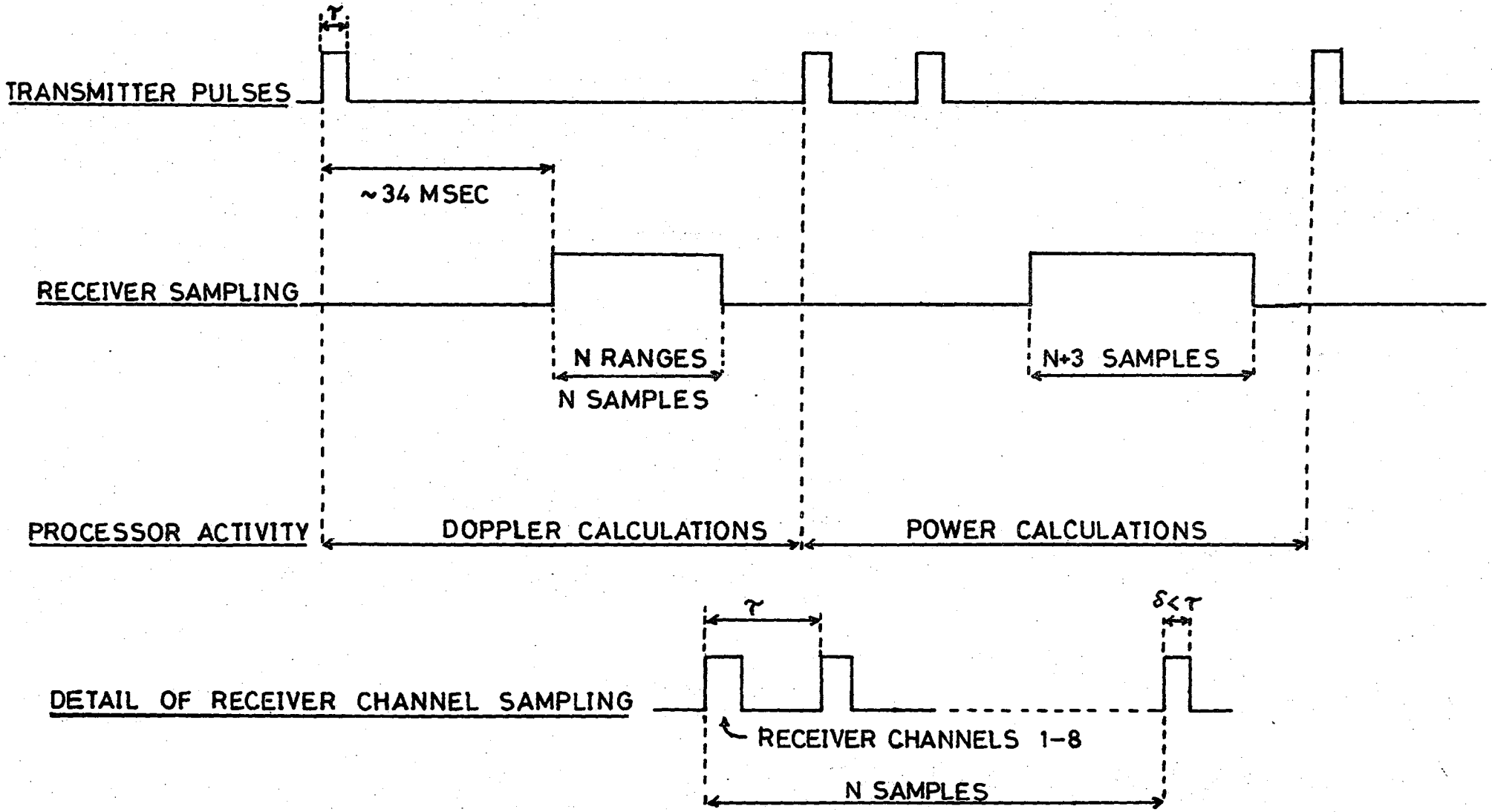

τ
TRANSMITTER PULSES
~34 MSEC
RECEIVER SAMPLING
N RANGES
N SAMPLES
N+3 SAMPLES
PROCESSOR ACTIVITY
DOPPLER CALCULATIONS
POWER CALCULATIONS
τ
δ<τ
DETAIL OF RECEIVER CHANNEL SAMPLING
RECEIVER CHANNELS 1-8
N SAMPLES

FIGURE 2.5

Chapter 3

The SABRE Design Philosophy

3.1 Error Checking

The Micronova system has been included in the SABRE radar computer system to perform concurrently tasks that, if performed by the Nova 3, would reduce the data throughput of the system. One such task is the monitoring of critical voltages and currents in the transmitter and receiver. This is important to ensure that faults developing in the system may be detected, and therefore corrected, as quickly as possible. In the STARE system faults that have developed in the transmitter, that have reduced the Effective Radiated Power (ERP), have remained undetected for periods of up to a few weeks. This is not surprising since at times the backscattered power is very low for long periods of time. To ensure this does not happen in the SABRE system the Micronova is responsible for measuring parameters such as the ERP for example. This quantity may be deduced from the analogue voltages and currents, in the transmitter, that are actually monitored by the Micronova computer. It is able to monitor 32 analogue channels and for each a course of action is prescibed should the voltages fall outside predetermined limits.

Having decided that some action is required upon detecting an abnormal condition, the Micronova can initiate a sequence of events to rectify the situation. This is achieved by a number of digital control lines that essentially provide the feedback to the transmitter and receiver. In the case of extreme failure a shutdown sequence may be initiated in an orderly manner, thereby avoiding cascading failures.

3.2 Interprocessor Communications

The communication between the two computers, the Micronova and Nova 3, is a very important sub-section of the whole system. Inefficient communications would lead to deterioration of system performance and possibly excessively complicated software. There are two communicating modes in the SABRE system, the first being the simplest and most common technique, that is, a 9600 baud asynchronous link. This high speed terminal link was used for debugging the system since it allows a terminal connected to the Micronova primary teletype, to talk via the Micronova secondary teletype, to the Nova 3 primary teletype. The software in the Micronova merely passes the characters from the primary to the secondary teletype and vice versa. This entirely software approach avoids having to change hardware around to talk to one or other of the computers.

The second communications technique is less common and is usually reserved for communications between a computer and high speed peripherals, such as a graphics or disk unit. DMA (Direct Memory Access) allows a very high-speed transfer of data, in this case 177 thousand words per second (KWDS/Sec), between the two computers. The advantage of this method, apart from the speed considerations, is that it requires no processor intervention after the transfer has been initiated. The computer-computer DMA interface exhibits a 1:1 linear mapping from one memory to another, however the start address of the transfer may be different in the two memories. The choice of two software compatible computers was important, since it is not only data

that may be transferred between the two machines by this method, but also programs may be directly transferred from one machine to the other. Because no software intervention is required, no complicated and time consuming protocol is needed for the two computers to initiate a communications sequence.

The DMA interface is controlled entirely by the Micronova, and so essentially the Nova 3 is acting as a non-intelligent peripheral as far as this interface is concerned. If radar parameters need to be altered then they may be sent straight to the Nova 3's memory without software intervention, by the Nova 3, similaly if current data is to be communicated along the telephone line it may be returned from the Nova 3 to the Micronova and subequently transmitted. The current program state of the Nova may be monitored by the Micronova merely by monitoring, via DMA, locations that are being altered as the programs progress, for instance the number of averages taken during the current integration period, or the current time. The important consideration is that these transfers may be achieved without stopping or interfering with the radar operation in any way, and so may easily be added to the STARE system without modification.

3.3 Modem Communications

The Micronova system has the ability to communicate with the telephone network via a modem interface. The ability to communicate between the analysis computer and the radar sites is a unique feature of the SABRE system. This enables any system failures to be reported by the Micronova immediately to the

analysis site. Radar data, concurrently being taken and processed by the Nova 3, may be transferred as a series of packets under the control of a software protocol that is responsible for ensuring the data integrity of the received information. Data is initially transferred from the Nova 3 to the Micronova, via DMA, and is then ready for transmission to the analysis site. Commands received by the Micronova are processed by the operating system to determine the course of action required. This enables the calling station not only to receive data and system parameters, but also to interactively modify the operating parameters of the system should this prove necessary.
The ability to not only receive real time data, but also to pass back previously recorded data that was stored on tape, has the advantage that it is not therefore necessary for tapes to be collected from the site on a daily basis as it is in the STARE system.

3.4 Bubble Memory

The bubble memory represents a part of the system implemented as a direct consequence of recent advances in technology. It is a 1 Megabit serial device that is ordered by the software into 64x1024, 16 bit words of memory for storing the operating programs, for both computers, and the most recent system state as determined by the Micronova. As the radar operates, the data from the error checking channels is continuously being written into a 32K word segment of the MBM (Magnetic Bubble Memory) which is acting as a circular buffer. The other 32K segment is used for storing the operating software

that is reloaded in the event of a power failure. The non-volatile nature of this type of memory enables the current and previous system state to be recovered in the event of a catastrophic failure simply by unplugging the MBM module and taking it away for analysis. In this way it is operating rather like the black box flight recorder in an aircraft. Also, because of its non mechanical nature it provides a very reliable form of non volatile storage that requires the minimum of maintainance. Disk units could not be used because of their susceptability to the environment, particulaly dust and temperature changes, and also because of the frequent maintainance that they require.

3.5 Power Fail

Semiconductor memory was used for both computers because of its speed, availability and easy maintainance. The big disadvantage as far as this type of experiment is concerned, is its volatility. The consequence of this is that provisi has to be made for the resumption of programs after power failure. Two solutions are implemented in the SABRE system and provide the means to restart the system under all conditions. The Nova 3 memory is equipped with battery backup to maintain the information in memory when power is removed. The Ni-Cad batteries that provide this are charged during the time that power is available, but will only maintain the information in memory for 3.75 hours after power is removed, if they were initially fully charged. This is a good solution to short power failures, but the system still needs to be protected in the event of extended power failures. No battery backup is provided for

the Micronova system and this computer is reloaded even after short power failures, from the bubble memory. This ensures that the Micronova has the operating software correctly installed in memory after power has been resumed. This overcomes the problem that happens occasionally with the STARE system, that the software becomes corrupted slightly presumably due to transients when power has been restored. Such memory corruptions as these have remained undetected in the past for considerable periods of time. Normally the Nova 3 will restart whilst the Micronova is being reloaded. The first test the Micronova performs is to check the state of the Nova memory by observing memory locations as described earlier. If there is some problem such that the Nova has not restarted correctly, the Micronova will map the entire contents of its 32K to the Nova 32K and restart the Nova program. The software for both machines always exists in both computers so that software integrity and memory errors may be detected by comparing the contents of the two computers memories.

3.6 Pulse Shaping

In the STARE system the bandwidth of the transmitted pulse was kept within the required 10KHz by providing a passive network filter to shape the raw TTL pulse. This was used to shape the pulse before transmission and remove the high frequency components. This approach is adequate but not ideal for two reasons. Firstly, these filters are not adaptive and they cannot be easily or cheaply changed to account for the changing characteristics of the output valves which tend to exhibit

frequency non-linearities with age. Also these circuits may produce unwanted resonances in the transmitter particulaly as the characteristics change with age. To counter these problems in the SABRE system the computer has control over the pulse shape which is dynamically alterable by the software. This allows much greater adaptability of this part of the system. A photograph of the shaped transmitter pulse is given in figure 3.2. Interference tests have been carried out at the site and have indicated that the spurious harmonic interference, normally associated with pulsed transmitter systems, is very small indeed. No quantitative measurements have, as yet, been made but the ability to tailor the shape of the pulse to the transmitter characteristics has not only been instrumental in achieving this low interference, but also in obtaining the maximum effective radiated power from the transmitter.

3.7 Tape Deck

The tape deck used was chosen because of the extra flexability that it could provide. Firstly it has a microprocessor-controlled diagnostics unit which allows the tape unit to test itself and isolate both mechanical and electrical faults, and secondly the formatting of the data is done by a hardware formatter unit. This last feature is very important because it allows the tape unit to be started by a two line program that will load the first 4K of data from the tape to the computer memory. The tape deck is a DMA device which has a DMA interface mounted inside the Nova computer. After power fail the registers inside this interface are all cleared to zero. The consequence of this is that the tape command is set to READ, the start address is set to 0 and the two's complement of the number of words to be transferred is 0. The two line program that initiates the single block read is given below.

```
Location 376    60122    NIOS 22
         377      777    JMP .-1
```

When the data is being loaded via DMA the processor is still executing locations 376 and 377. Therefore when these location are overwritten by the data on the tape it should be arranged such that it is overwritten by executable code such as:-

```
Location 376        JMP 100
         377        JMP 100
```

This would cause a CPU jump to location 100 which would have been previously loaded and would be therefore unchanging at this time. From this location would be a small program to load the next block and all subsequent blocks from the tape deck until all 32K had been read from the tape. By this method from the initial two word bootstrap all 32K of memory would have been loaded from tape. Since the bootstrap may easily be entered through the front panel switches it is a very fast way of restarting the Nova by hand.

This technique, made possible by the hardware formatter on the tape deck enables the Nova 3 to be autonomous of the Micronova should the latter computer fail, which is, of course, a possibility. If this type of deck had been available during the conception of STARE, the paper tape readers would not have been required as the software could have then been loaded from tape.

3.8 Operating Software

The usual approach to the operating software for a remote experiment is very much a stand alone system. This comprises of a program specifically designed to control the hardware in a particular way with usually no consideration given to flexability. This approach may be justifiable since usually a new program may be developed at a separate data analysis facility and subsequently shipped to the site when required. An approach such as this has worked at the STARE facility for a number of years, but it has some drawbacks. It is usually impossible to effect any significant changes to the software once the system has been installed at the site, especially since the Nova 2 computers lack

any form of monitor. Secondly, the delay in preparing new programs and implementing them at the site may be considerable due to the large distances in both the STARE and SABRE facilities between the site and analysis facilities.

To overcome these problems software has been included in the SABRE system which allows not only easy viewing and modification of existing software, but also the creation of new software at both an assembler and a high speed language level. To allow for the lack of a mass storage unit such as a disk, the language is entirely core resident and provides an interpretive monitor as well as a compiler and assembler.

The presence of such a system allows terminal communication at the site with either computer, enabling test programs to be compiled and run to test hardware and interfaces for either initial testing, or fault isolation and debugging.

3.9 System Clocks

It has been found that the clocks used in the STARE system have been very susceptible to voltage transients that sometimes modify the time in an unpredictable manner. Another problem is that they can only be set by hand and this has made synchronisation of the sites in Norway and Finland very difficult. In the SABRE system a new clock interface was designed. With the aid of the communications link already described, it is possible from the analysis site to not only check the system time, but also to alter it if required. This is an important consideration because the time accompanies all data,

whether it is written onto the tape, or transmitted to the analysis site.

3.10 Conclusion

In the SABRE system with the Micronova being operated as a front end processor, but with intimate (so-called tightly coupled mode) communication between the processors, the system will still operate even though the hardware may be faulty. For example, failure of the Micronova or the DMA communications board would not be catastrophic, indeed the Nova would still continue to run and take data. Failure of the Nova, transmitter, or receiver would be reported by the Micronova enabling at least the fault to be identified quickly. The SABRE system has been designed to not only rectify the weaknesses inherent in the STARE system, but also to provide a very much enhanced mode of operation.

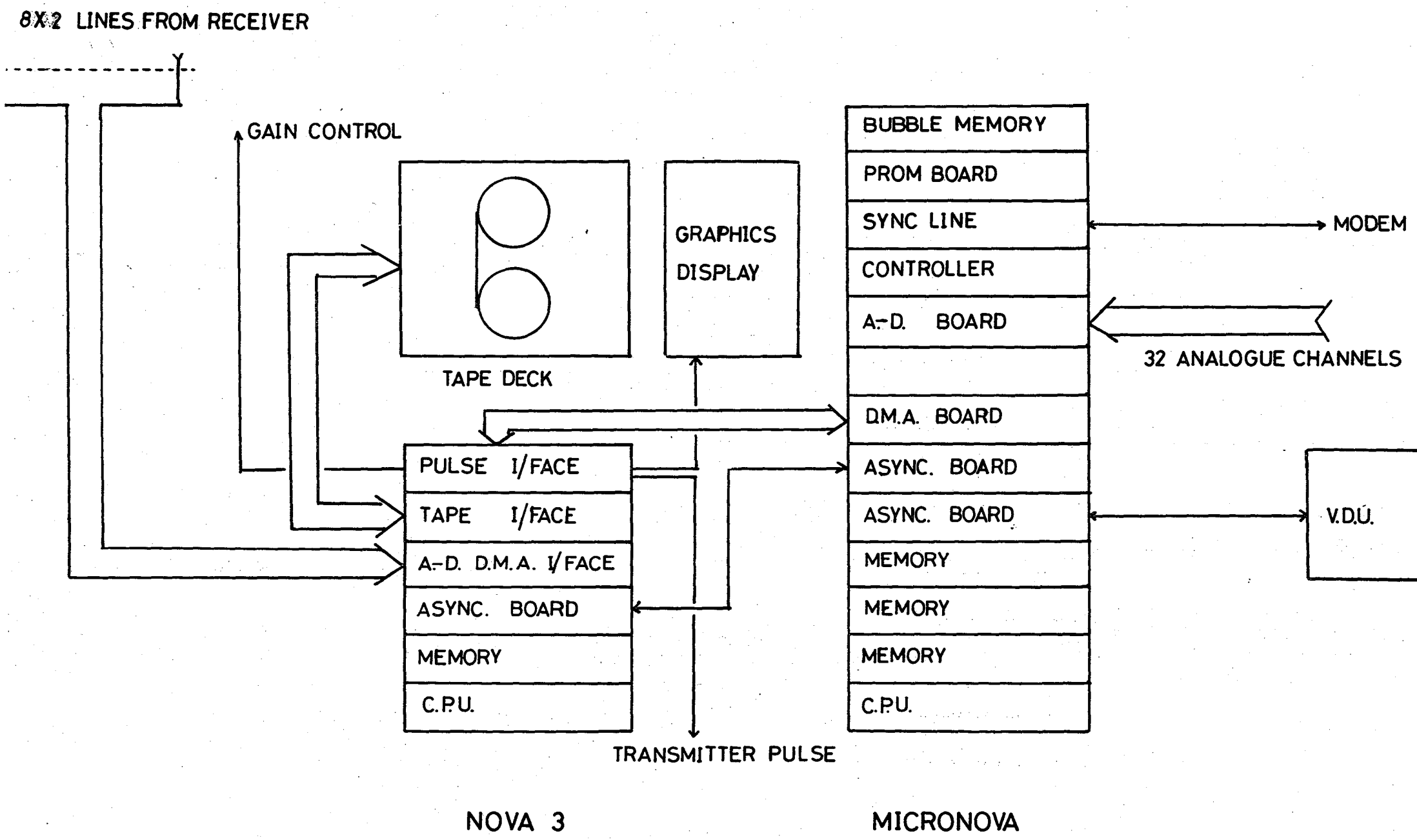

8X2 LINES FROM RECEIVER
GAIN CONTROL
TAPE DECK
GRAPHICS DISPLAY
PULSE I/FACE
TAPE I/FACE
A-D. D.M.A. I/FACE
ASYNC. BOARD
MEMORY
C.P.U.
TRANSMITTER PULSE
BUBBLE MEMORY
PROM BOARD
SYNC LINE
CONTROLLER
A-D. BOARD
D.M.A. BOARD
ASYNC. BOARD
ASYNC. BOARD
MEMORY
MEMORY
MEMORY
C.P.U.
MODEM
32 ANALOGUE CHANNELS
V.D.U.
NOVA 3
MICRONOVA

FIGURE 3.1

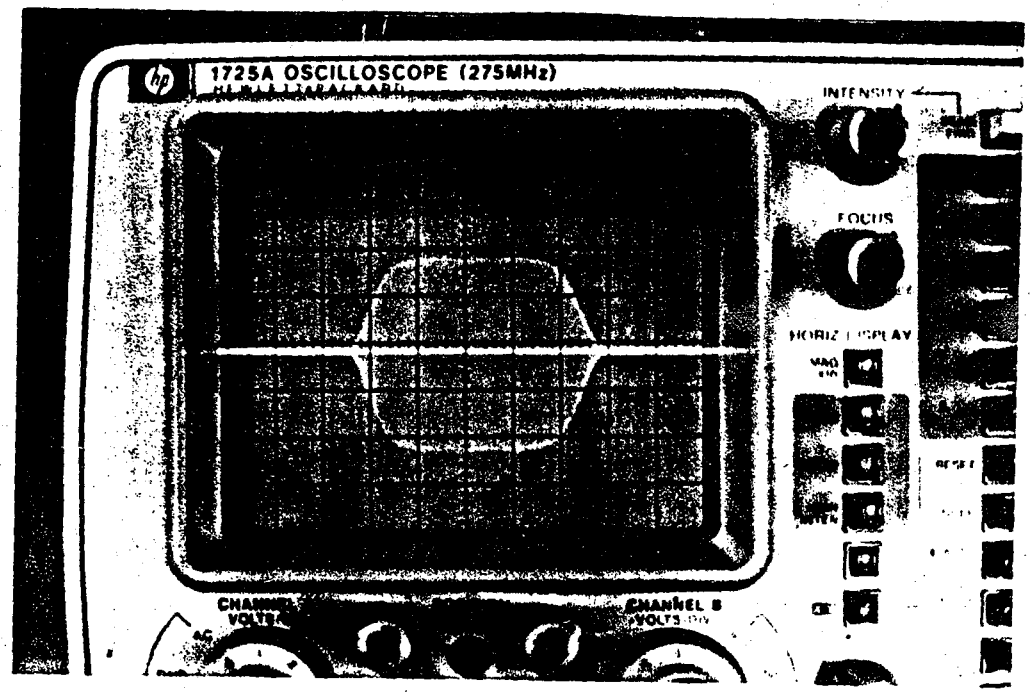


FIGURE 3.2

Chapter 4

The Computer Operating System

"The limits of my language
stand for the limits of my world."

Wittgenstein - Tractacus Logico Philosophicus.

4.0 Introduction

FORTH is a computer language that has, over the last 5 years, become well known and used in a wide variety of process control and instrumentation applications. FORTH was developed in the late 1960's by Charles Moore and its advantages have been subsequently outlined in many papers (Rather, 1977; Hicks, 1979; Moore, 1980). Its versatility to allow easy access, for the programmer, to the hardware on which it is implemented, coupled with its high speed and inherent small size has made it a popular tool for software development worldwide. Described here is a second generation FORTH, called ROS, (Radar Operating System) which maximises the advantages of FORTH, whilst making full use of the architectural features of the machine on which it is implemented.

4.1 Description

The SABRE operating system, (ROS) was designed to provide a convenient high level communications interface between the operator and the machine. One of the constraints of the hardware for SABRE, as indeed it is with many real time stand alone systems, is the lack of high speed mass storage facilities. The main reason for this is that at the present time units such as disks require continuous maintainance to ensure their continued operation. It is with this in mind that ROS has been designed to be totally memory resident. However because of its extensive modularity it can achieve all the functions of a compiler, interpreter, assembler and editor. This maximal system together

with all the application programs occupy less than 16K words of memory ! The small size is one of the important features of ROS and indeed a well thought out program may actually occupy less memory space than a badly structured equivalent assembler program.

ROS is essentially an interpreter with the compiler being merely a special case of this mode. ROS prompts the operator when input is required with the "$" character and then expects to see an input line of ASCII characters. This input line must conform to a syntax that may be described by the Backus Naur representation given overleaf. Here general names or forms are enclosed in brackets such as **(LETTER)** , with the form given in capitals. Explicit ASCII characters are enclosed in double quotes thus ":" refers to the actual character : . For example the form **(M6800)** may be equivalenced to the characters "computer" by the following :-

(M6800) := "computer"

The symbol := is the equvalent of "is defined to be", and the vertical line, " " is the logical operator OR. The repeat statements **k=1,N** refer to the previous form enclosed in brackets repeated from at least once, to **N** times, where **N** in this case is essentially any integer value.

(LETTER):=A,B,C,D,E,F,G,H,I,.....Z,a,b,c,d,e,f,g,h,i,....z

(DIGIT):=0,1,2,3,4,5,6,7,8,9

(OTHER):=!,",$,3,&,' etc.

(DELIM):= (LETTER) < (DIGIT) < (OTHER) but is normally a space " ".

(CR):= 15 ASCII carriage return base 8

(WORD):=((LETTER) < (DIGIT))k=1,N

(SENTENCE):=((DELIM))k=0,N ((WORD) (DELIM))k=1,N (CR)

The repeated forms may alternatively be expressed as recursive forms such as :-

(SENTENCE):=(WORD) (DELIM) (SENTENCE)

The (WORD) constitutes a command that ROS will try and interpret and a sentence is merely a sequence of such words separated by at least one delimiter. Each word in the sentence would then be interpreted in a sequential manner as it is entered in the command line from left to right after a carriage return has been entered. The following illustrate some examples of valid word and sentence structure.

1 w W1 HELLO 23 are valid words syntactically because they are composed of printable ASCII characters.

This IS thel FIRst TlesT would be a valid sentence

syntactically.

So far ROS has been shown to be able to take a word isolated from other words by a suitable delimiter, and in some manner to be described later, execute the routine in memory to which the word corresponds. In so far as this goes ROS is an interpreter, but there is the capability to add new definitions which are constructed by reference to previously constructed definitions. This may be likened to a very extensive macro facility, which indeed it is, but it is also much more. ROS is a structured language, similar to Pascal, in that it has no GOTO or statement labels. This structuring is important for program correctness and maintainability since the logical flow of the programs are well defined and contained within an easily testable sub unit, namely the short ROS definitions.

ROS object code, that is the directly executable part of the definition, is extremely compact. The reason for this is that no matter how much a particular definition does, it is compiled in at most two words, as a subroutine reference. The bigger the program the greater the memory advantage since the heirarchical nature of programs allows increasingly powerful and application targeted programs to be built up. The kernal of ROS, which is written in assembler, contains all the primatives required to establish the compiler and terminal communications. The subroutines of the kernal are listed fully in appendix 1.

The fact that ROS is entirely self contained in memory means that there is no need to invoke editors, compilers and loaders separately since these are all memory resident. The trade off here is that there is no parser, in the conventional compiler sense, because all words are encountered and processed in a left to right (L-R) manner completely in one pass. A second optimisation pass is also included and will be described later. All arithmetic operations have therefore, to be pre-parsed and that means already in Reverse Polish notation. Thus the following arithmetic operation in BASIC or FORTRAN :-

I=A*(B+(C-D)*2)

would have to be entered as :-

C D - 2 * B + A * I !

The I ! is the ROS equivalent of equating a value to a variable.

Thus the mathematical operations are restricted to Reverse Polish integer applications only. This does not prove a restriction when a control application is required, as in SABRE, but is usually unsuitable for large 'number crunching' applications. The size of ROS has been kept to a minimum by tailoring the available functions for the application, although the generality is such that floating point operations could be defined as required.

4.2 Compiler Operation

The creation of a new definition is initiated by executing an already existing definition, that modifies the dictionary structure to incorporate the new definition. In this sense the first part of the compiler is merely interpreting and executing a current definition. This definition is invoked by a word comprising of a colon character. The following BNF shows a new definition called SABRE being created.

(NEW) := (DELIM) "SABRE" (DELIM)

The first part of the new definition is the name, which is equated to the form (NEW) and is comprised of the ASCII word SABRE separated by delimiters.

":" (NEW) (SENTENCE) ";"

Hence ":" initiates the compile mode whilst ";" terminates it. The logical consequence of this is that during compile mode the interpreter still tries to identify the succession of words, but instead of executing the routines they represent, passes control to a small compile program that compiles the word currently referenced. This compile program is an extension to the code that decides whether a valid definition is to be executed or compiled. This decision is taken on the basis of comparison of two flags; the machine state flag and the definition precedence flag. Figure 1 indicates the course of action to be taken as a result of this comparison. From this it can be seen that a definition of precedence 2 will always be executed even during a sentence that is currently being compiled.

This is very important because it allows a definition to modify the course of compilation whenever it is encountered. So for example, a high level definition may be written that has a precedence of 2 and when executed moves the current pointer to the next free place in memory, N words further along. This would then reserve essentially an array during compilation in whatever definition it was referenced. This is a very simple example, and the compile time directives are shown more fully in appendix 2, but this example illustrates the power and flexibility of the language. The fact that the user can modify how his programs are compiled is a unique feature in any language !

The usual course of events however, is that a definition would have a precedence of 0. During interpretation the system state would also be 0. Thus when a definition has been found it would be executed as indicated in figure 1. When ":" is executed one of its primary functions is to increment the system state thus:- STATE=STATE+1. It also isolates the next (WORD) and enters the first four characters and the length, with an implicit precedence of 0, into the next free place in the dictionary according to the format in figure 2. Because the state is now greater than the subsequent definitions of precedence 0 the latter would be compiled. From the previous discussion it is clear that during the compile or execute phase two distinct conditions can arise. Firstly a referenced word has been previously defined. If this is the case compilation or execution may proceed. Secondly, if a word is undefined, ROS will try to interpret it as a number in the current base. If this is

successful the value will be pushed onto the stack, otherwise an error condition exists. This will force the computer to type the offending word in the following format:-

(WORD) "?" CR

The same error message will ensue if an error condition exists in either the compilation or execution mode.

4.3 Dictionary Structure

Clearly then, one of the basic functions is to be able to look for a given word in a set of currently defined words. To achieve this the existing words form a dictionary or linked list. That is, all definitions comprise, as indicated in figure 2, of a 4 word header, a link address, the executable code, and the associated addresses. The last definition is pointed to by an address stored in a zero page location, and so is available to the routine which is responsible for searching the dictionary. The Pth definition has a link address which points to definition P-1, which subsequently points to definition P-2 and so on. A link address of zero indicates the end of the dictionary. Clearly dictionary entries can be of variable length but the link address can always be found at offset N+4 from a header at location N. This makes it very general and the routine finding the dictionary entry can treat all entries in the same manner since all it is interested in are the characters stored in the header and the link address to the next definition. Figure 2 shows the 4 word header in detail indicating the four characters that are stored to identify the definition.

The consequence of this is that uniqueness of a definition entry is limited by the first four characters and the length of **(WORD)** . This is an improvement over FORTH which looks at the first three characters and the length. The length is stored in word 1 of the header, as indicated in figure 2, and occupies the first byte of that word. Thus names are limited to 256 characters in length but this has not proved to be a major restriction ! Care should be taken when defining new definitions since the following names would be equivalent:- SABRE1 SABRE2 . They both have a length of six characters and the first four characters in each case are SABR. If SABRE1 was defined after SABRE2 all references to both names would result in the execution of SABRE1. This is because the dictionary is searched from the latest entry towards the first entry until a match is found.

Word 4 contains a null for future expansion and will either be used to extend the number of characters to 6, or to contain a parity check over the definition. This latter feature would make single bit errors which are typical of memory faults, easily isolated before a definition is executed with possibly catastrophic results. The 5th word of the header contains the link address back to the start of the previous definition. This enables easy searching of the dictionary but is by no means the only possible construct. A complete dictionary with definitions is shown in figure 3.

4.4 Stacks

ROS is an entirely stack oriented language in that all numeric references are passed to the operational stack to be operated on. As an example consider the input line :-

```
4 3 +
```

The number 4 is entered onto the operational stack first, followed by the number 3. Remember that these numbers are first compared to the dictionary entries to see if there is a match and are then interpreted as numbers, subsequently being placed on the stack. The operation + is defined to pop two numbers from the stack (3 first then 4), perform the addition then push the result back onto the stack. In this way complex operations may be performed without worrying about storage allocation although it is up to the programmer to achieve correct computation by considering what elements on the top of the stack will be modified by the current operation.

There are two system stacks; the DO...LOOP stack and the machine stack. The DO...LOOP stack is used to hold LOOP variables during a LOOP operation. By using a separate stack for these operations nested LOOP parameters may be accessed very quickly. This is another improvement over FORTH, because in the FORTH systems the DO..LOOP pointers are held on the machine stack and are typically interspersed with the return addresses also held there. There is therefore an overhead imposed in retrieving the LOOP parameters. By holding the parameters on a separate stack the access time, particulaly of nested loops, is greatly reduced.

The machine stack is used primarily to retain subroutine return addresses thus allowing a high degree of nesting which is one of the main features of ROS since essentially each definition is a self contained subroutine.

The machine stack uses the hardware stack facility on the Nova, so all push operations in assembler e.g. **PSHA 3** implicitly specify the machine stack. The operational stack pointer is kept in accumulator 2 of the CPU which means that it is always readily accessible to a user program. The DO...LOOP stack pointer is kept as a zero page variable and may be accessed. The subroutine processing on the Nova allows no nesting at all. The return address after a subroutine call is placed in accumulator 3 thus destroying the previous contents. Nested subroutines therefore need a formal way of saving all the return addresses required, and this is normally done by pushing them onto a stack. Thus, contained within the body of a particular definition, the format required to achieve subsequent indefinite nesting of that routine is as shown in figure 2. Because the Nova does not implement a SAVE/RETURN sequence directly, two instructions are required to exit a subroutine, one to recover the address and one to execute an unconditional jump.

4.5 Compiler Optimisation

When in compile mode, the interpreter having located the **(WORD)** via a dictionary search passes control to the section of code responsible for extending the dictionary. Therefore the information that is available about a definition is the start

address of the code that it represents. A **JSR@** relative indirect jump to subroutine instruction is stored in the next free dictionary location and the address just found is placed in a compile table. This process continues until all words in the current **(SENTENCE)** have been processed.

Consider the compilation of :-

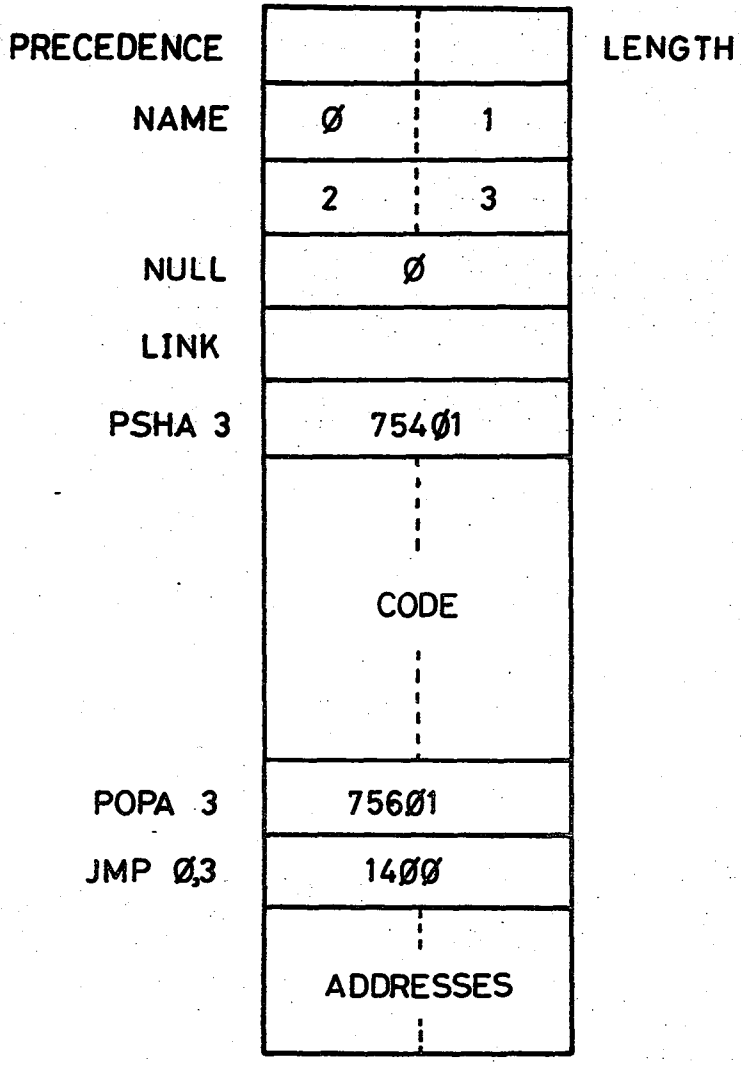
: SABRE CRLF CRLF TITLE ;

The sequence of events is as follows :-

(1) The system state flag is initially 0, that is interpret mode.

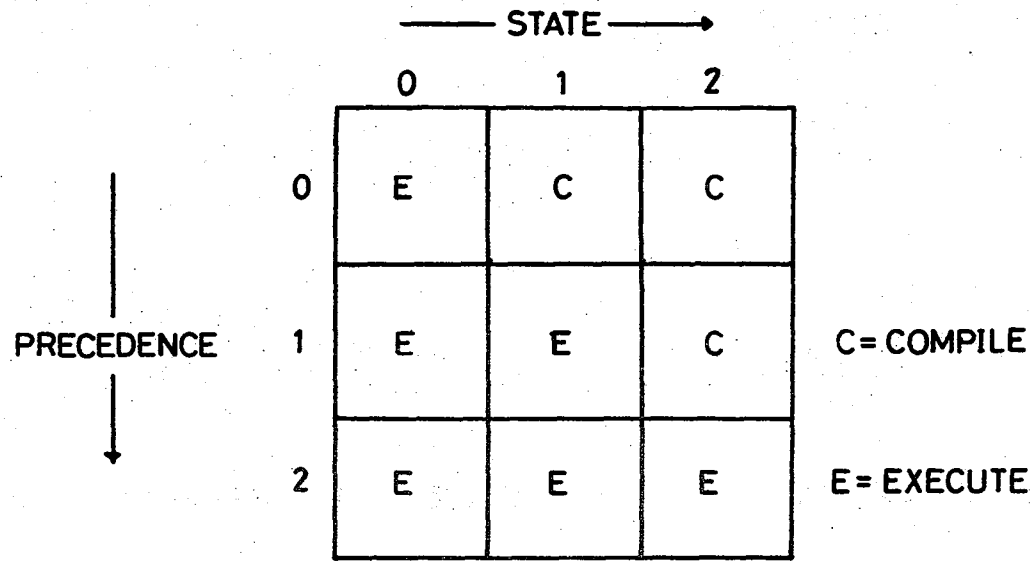
(2) ":" is found and executed, this puts the system state to 1 and looks for the next word which is "SABRE", instals the first four characters SABR in the dictionary and inserts the machine code for the instruction PSHA 3 into the dictionary.

(3) "CRLF" is found, the precidence is 0 and is less than the state which is now 1. Thus this definition is compiled into the current dictionary position by inserting the skeleton code for **JSR@** but with no address as yet. The start address of the routine "CRLF" is placed into a compile table.



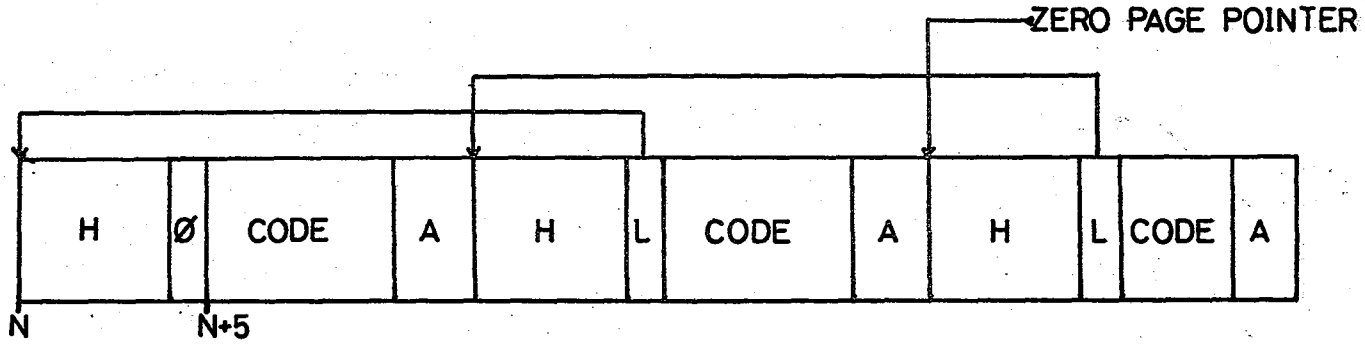
DICTIONARY ENTRY FORMAT

FIGURE 4.2



MODE TABLE

FIGURE 4.1



LINKED DICTIONARY

FIGURE 4.3

Chapter 5

The System Software

5.0 Introduction

The software for the SABRE radar controller is divided between that which runs on the Micronova and that which runs on the Nova 3. The software will be discussed with reference to the machine on which it is implemented so no confusion should arise. In this chapter is presented a detailed discussion of the important routines. Not all routines listed in appendix 3 are discussed as it is felt that some are primarily self explanatory, particulaly since all routines are highly commented. When a reference to a particular subroutine is made it will be called by the name that would invoke execution of the routine via the operating system ROS.

5.1 Micronova Programs

The logical flow of operations carried out by the Micronova is shown in figure 5.1. This flowchart represents the routine MAIN which is responsible for calling all the subroutines required. These subroutines will be discussed in the following sections under the general description of the functions that they perform.

5.1.1 PROM Software

The program contained in PROM (Programmable Read Only Memory) for the Micronova is a bubble memory bootstrap program. On power fail the processor vectors to location 74000 (base 8) which is the start of the PROM program. This address was set by a number of alterable, or "strappable", options available on the central processor (CPU) board. The available PROM space is the top 2K of memory, although only a small proportion of this is used. Indeed the PROM board may be sited anywhere in memory but the top 2K was used since this would allow a disk based operating system, DOS, to be used (Data General, 1981a,1981b). Initially software development was carried out on the Micronova computer using DOS and a twin floppy disk unit.

The PROM program sets up all the bubble memory parameters, such as the error correcting level and the number of FSA channels to use, which is always 2 (Intel, 1981a,1981b,1981c). Initialisation is then instigated and the software waits for successful completion. At this time a page can be read. No attempt is made, whilst reading a page, to assemble the byte oriented output into the 16 bit words that constitute a full memory location. The 64 bytes that are read from a page on the bubble memory are buffered into an area of RAM starting at location 67777 (base 8). After a full page has been read these bytes are assembled into words and relocated to their correct position in memory, after which another page is read. The system memory is stored, in the bubble memory, from location 22 to location 67700. This data occupies the bubble memory from page 0

onwards. After all the data has been read from the bubble memory, the PROM program executes an indirect jump through location 25 which contains the start address of the Micronova operating program. The operating system, ROS, that is loaded by this method contains definitions, as described in chapter 4, to save a "core image" in the bubble memory. In fact to initially save the operating system in the memory, ROS was loaded from disk and then used to save itself ! By changing the contents of location 25 and saving the memory contents again (program ALL-MEM in appendix 2), any desired program may be executed after a power failure.

5.1.2 Data Acquisition

The subroutine that reads a block of data from the error monitor analogue to digital converters may be tested by typing DBLOK from the terminal, which is the ROS name for that routine. Two channels are read simultaneously since each A-D conversion is only 8 bits, two channels are read in one 16 bit word. This data are stored in a zero page buffer of 16 words.

To be able to detect error conditions, the A-D values read are compared to a small data base. This data base contains 32 entries, one for each channel and each entry is 4 words long. The first word is the maximum allowed value for that channel as an unsigned 8 bit integer. The next entry contains the minimum allowed value. The third entry contains the start address of a subroutine to be executed should the maximum allowed value have been exceeded. Finally the fourth entry contains the start address of a subroutine to be executed should the minimum allowed

value have been surpassed. Thus for each of the 32 channels a course of action is open should an error condition occur.

The data base comparison routine is called DCOMP, and should be called after DBLOK. If an error condition occurs, however, the routine DBLOK will be called by DCOMP, and DCOMP executed recursivly. This ensures that once a course of action has been taken then the effect will be monitored continuously.

The following voltages on the transmitter and receiver are monitored:-

(1)..Driver Power out

(2)..24 Volt Supply

(3)..Driver Power input

(4)..Driver Reflected Power

(5)..Driver EHT Voltage

(6)..Driver EHT Current

(7)..Driver Heater Voltage

(8)..Driver Air Flow

(9)..P.A. Power Out

(10)..13 Volt Supply

(11)..R.F. Sense Inhibit

(12)..P.A. EHT Voltage

(13)..P.A. EHT Current

(14)..P.A. Heater Voltage

(15)..P.A. Air Flow

(16)..RX Auto Mode Status

(17)..RX Power Supply Status

5.1.3 Power Fail

Having restarted after power fail the Micronova is able to monitor the status of the Nova 3 and determine the course of action to be taken. The routine PFAIL looks at the memory location in the Nova 3 that holds the number of averages of pulse sets when the radar is running. This is done by using the DMA board that connects the memories of the two machines together. If after a time of 100 seconds the value does not change the Micronova will reload the memory of the Nova with the operating program, again by DMA. The Nova is restarted in the following manner.

(1) The Nova memory is filled with 0's. This ensures that the Nova CPU is in a known condition, as execution of 0 as an instruction is jump to location 0 (JMP 0).

(2) The memory is reloaded from location 1 upwards, by DMA until all the operating software is present. The CPU at this time is still cycling at location 0.

(3) When all programs are present an indirect zero page jump instruction is stored by DMA into location 0. This causes the CPU to find the start address of the program in zero page and commence execution.

5.1.4 Main Program

Initially after power has been restored this routine is executed and sets up the interrupt service address in location 1 for the synchronous communications interface. The power fail

routines are then executed and the system is returned to an operational state. Each of the programs concerned with the voltage monitoring and the bubble memory are called in turn. The sequence is repeated as indicated in figure 5.1.

This program therefore contains the overall flow logic of the Micronova programs, and being self contained it is easy to add other subroutines and call them from this program, should the need arise.

5.2 Nova 3 Programs

The Nova programs, in a similar way to the Micronova programs, are controlled by an overall controlling routine, in this case called MRUN.

5.2.1 Power calculations

The raw data from the receiver is read into an array in memory that is typically 400 words long. The length of the array is determined by the number of ranges under consideration. Usually 50 ranges are used, with 1 measurement for each channel per range. This gives 50x8=400 measurements. The data are read in with the high order byte being the real part of the quadrature signal, and the low order byte the imaginary part of the quadrature signal. For each element in the 400 word array the expression is calculated as given in chapter 2.

After this calculation has been performed the result is added to previous calculations for the same range and the accumulated sum stored in a second array. In this way a

continuous sum of the backscattered power is kept for each range. At the end of each integration period this power is averaged by dividing by the number of pulse sets and subsequently the arrays are written onto tape.

5.2.2 Doppler Calculations

For a double pulse separation of 300 microseconds the raw data array is 424 words long. Since one sample is taken per lobe every 100 microseconds, the 300 microsecond pulse separation corresponds to 3 words in the raw data array. To achieve 400 autocorrelation products the array needs to be:-

400+8(Lobes)*3(Extra words per lobe)=424 Words long

This is because, to perform the complex product, two raw data words are used separated by the number of words corresponding to the double pulse separation. The double precision products are accumulated in a second array which is subsequently averaged by the routine AVER at the end of the integration period.

5.2.3 Real Time Clock Routines

These subroutines are responsible for controlling and reporting the local system time. The RTC (Real Time Clock) is read by the routine CTIM which controls the actual interface. ETIM merely takes the time present in the zero page buffer, TBUF, which is filled by the routine CTIM, and calculates the end integration time, which is placed in the buffer EBUF. All the hour and day boundaries are adjusted by ETIM should the need

arise. If the end integration time crossed an hour boundary and end of file flag, EOF is set so that an end of file marker may be written to the tape. Routine TIUP reads the current time using CTIM and compares it to the time in ETIM. This is used to monitor the integration progress and call the appropriate routines when the period is finished.

5.2.4 Tape Routines

The subroutine TAPE is called to form the buffer for writing to the tape. The system parameters are copied from various zero page locations and are written at the start of each tape record. These parameters include the operating parameters for the radar and the start of integration time for that particular record. Then the averaged power samples, and finally the Doppler velocities are written.

The subroutine TWRIT writes the tape buffer, formed by TAPE, by controlling the tape interface. TWRIT also writes the end of file record should an hour boundary be crossed.

5.2.5 Interprocessor Software

The routine COMM is executed if a flag is set in the zero page location MICRO. This enables the Micronova to tell the Nova to perform facilities such as rewinding the tape or reloading the pulse sequence. This is necessary because the Micronova may load another pulse shape into the Nova's memory, but the Nova then needs to transfer it to the device interface using the routine PSEQ. The flag MICRO is tested every integration period and depending on the bit pattern in that location one or more

routines may be executed. One of the most important of these is that the system time may be reset remotely from the analysis site via the teleplone communications link. The time would be sent along the telephone line to the Micronova, then transferred to the Nova by a command issued to the Micronova. This command would execute the transfer routine and operate the DMA interface. Finally, another command sets the appropriate bits in the location MICRO and transfers that word, also by DMA, to the Nova. The Nova would then execute the routine COMM and would be directed to execute the routine to change the time in the device interface.

5.2.6 Power Fail Routine

The routine FAIL is executed as power is failing, and ensures the power fail auto restart integrity by writing JMP @ 300 into location 0. When power is returned the computer starts executing from location 0 and will thus jump to the address contained in location 300 which is the start address of the main program.

5.2.7 Nova Main Procedure

Similar to the Micronova, and shown in figure 5.2, this routine is responsible for calling all the other routines that constitute the logic of the program. Initially the interrupt vector is set for the program FAIL and the interfaces are reset. Storage for the program is calculated by the routine STOR which allocates the storage for the data arrays that will be required. This storage depends on the number of beams selected, the double

pulse separation and the range selecetd. The following equations indicate how the array storage required is calculated.

No. of power samples (PSAM) = No. of beams (NBEAM) x No. of ranges (NRANG)

No. of Doppler samples (DSAM) = Doppler pulse gap (PGAP) x NBEAM + PSAM

The transmitter and receiver are restarted after a power failure in the following manner.

(1) The receiver is pulsed to clear all the gating logic that protects the front end from being active and therefore saturated, during the transmit phase.

(2) The transmitter is then turned on but put into inhibit mode to reset all the internal timers.

(3) A wait period of 150 seconds is then established to allow the transmitter to run up to full operating power before pulsing commences.

(4) The program now reads the RTC and waits until it shows zero seconds. This ensures that the integration periods, which are a multiple of 10 seconds, will always return to minute boundaries. In this way the two sites may be synchronised after a power failure at one or both of the sites. This is of course done within the accuracy of the system clocks.

(5) The sense inhibit line is now taken low and this starts the internal timers that would turn the transmitter off if no pulses

are sent for any period greater than 250 milliseconds.

(6) Just before the program calculates the averages at the end of the integration period, the sense inhibit line is again taken high, thus clearing the timers in the transmitter since there are no pulses sent to the transmitter while the program calculates the averages and forms the tape array. This time is in excess of 250 milliseconds.

5.3 Conclusion

All of the software described in this chapter was written in assembler code to achieve the maximum possible speed. For the Nova 3 computer this is particulaly important since the number of single-double pulse sequences during the integration period is directly related to the time required to perform the power and Doppler calculations. Significantly, however, by incorporating the routines into the operating system as a set of separately executable modules, or definitions, these routines are able to be tested independantly of each other, or as a group. Being able to do this has greatly aided the on-site testing of the software, before using it to process 'real data'.

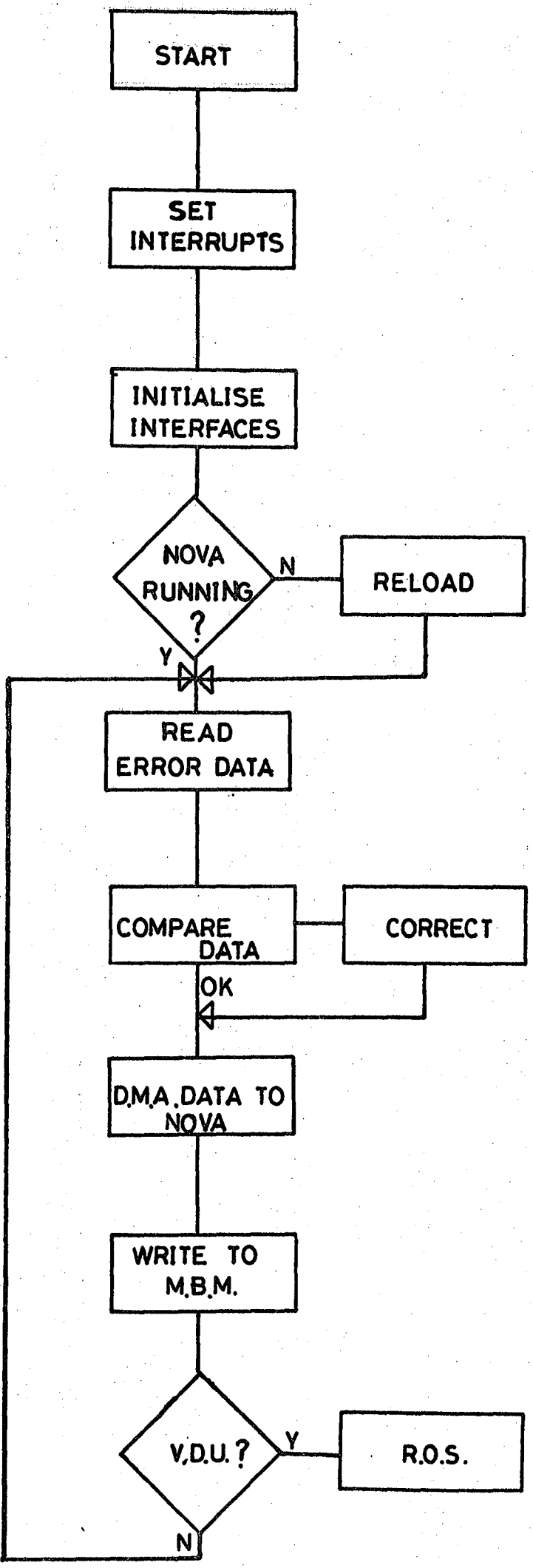

START
SET INTERRUPTS
INITIALISE INTERFACES
NOVA RUNNING ?
N
RELOAD
Y
READ ERROR DATA
COMPARE DATA
CORRECT
OK
D.M.A. DATA TO NOVA
WRITE TO M.B.M.
V.D.U. ?
Y
R.O.S.
N

FIGURE 5.1

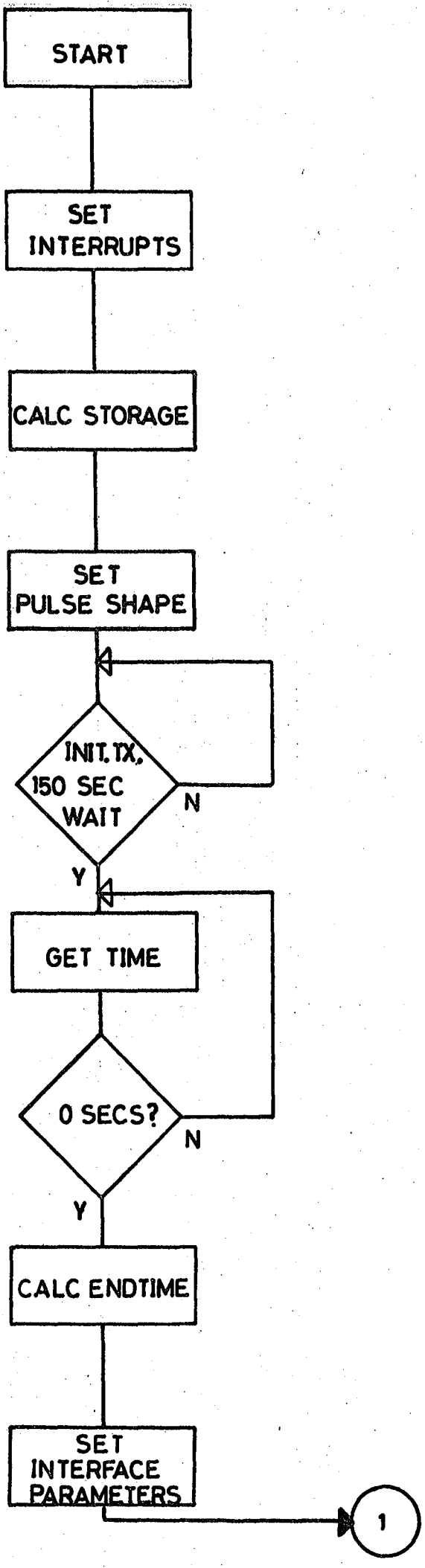
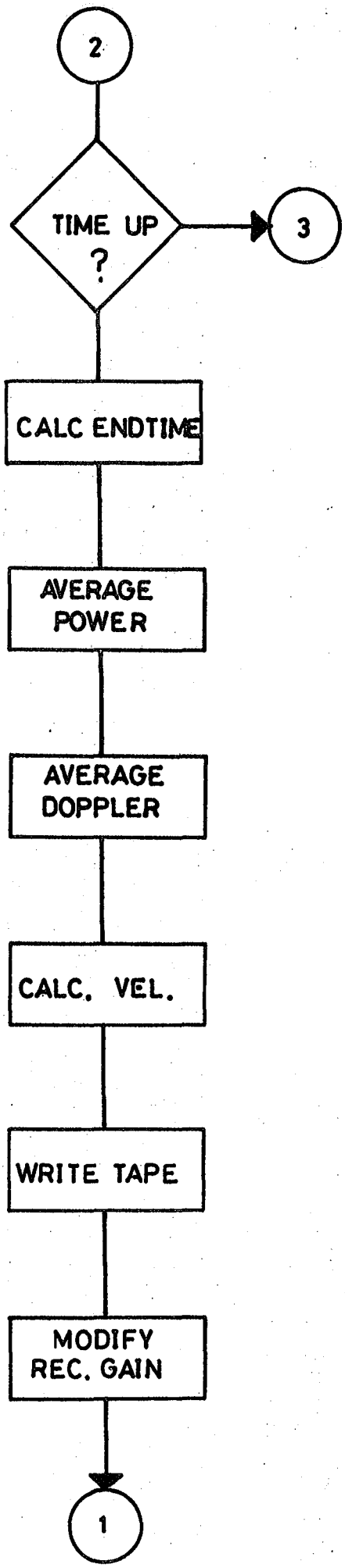
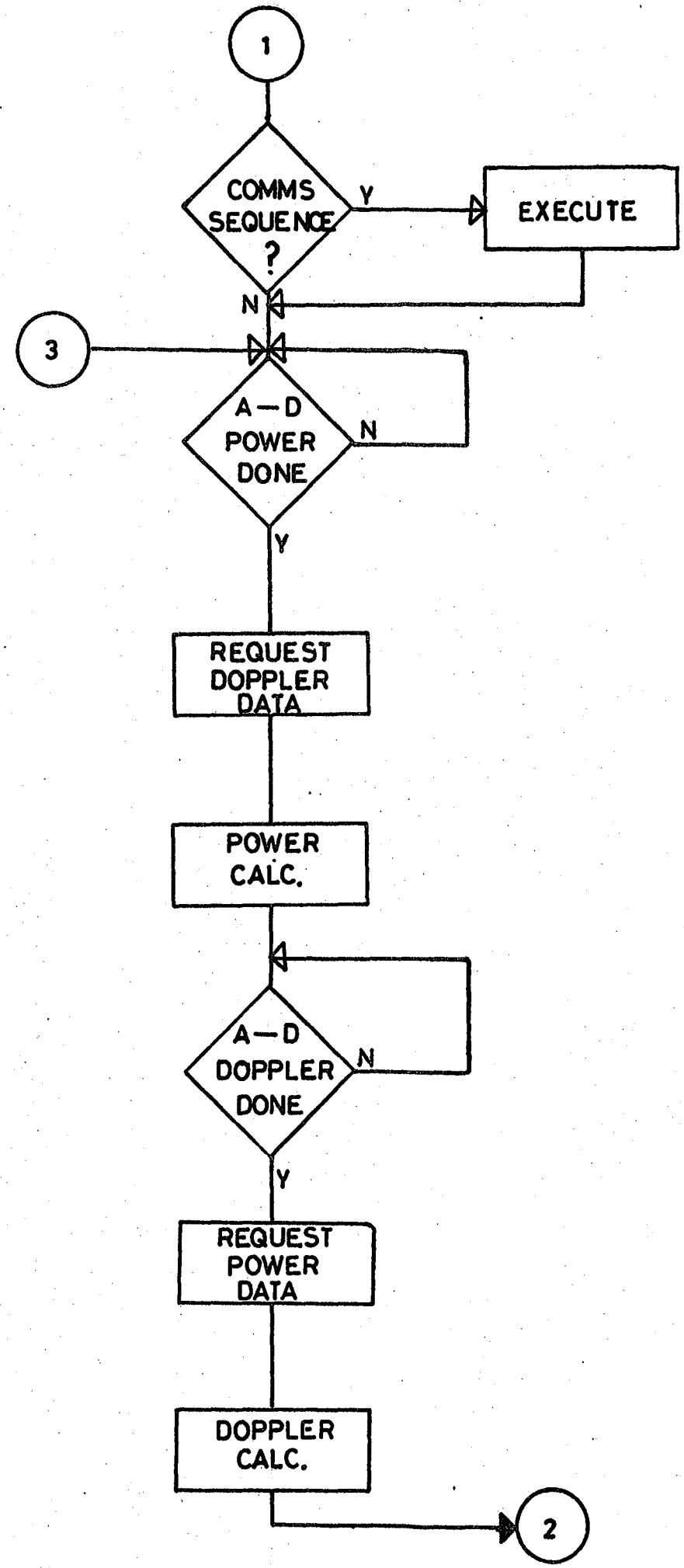

START
SET INTERRUPTS
CALC STORAGE
SET PULSE SHAPE
INIT. TX, 150 SEC WAIT
N
Y
GET TIME
0 SECS?
N
Y
CALC ENDTIME
SET INTERFACE PARAMETERS
1

FIGURE 5.2


2
TIME UP ?
3
CALC ENDTIME
AVERAGE POWER
AVERAGE DOPPLER
CALC. VEL.
WRITE TAPE
MODIFY REC. GAIN
1


1
COMMS SEQUENCE ?
Y
EXECUTE
N
3
A—D POWER DONE
N
Y
REQUEST DOPPLER DATA
POWER CALC.
A—D DOPPLER DONE
N
Y
REQUEST POWER DATA
DOPPLER CALC.
2

Chapter 6

Remote Communications System

6.0 Introduction

The last decade has seen many fundamental advances in the field of computer communication systems. The establishment of both hardware and software protocols has meant that international computer networks have become both technically and economically viable. However, because of the simultaneous research into the development of computer networks, a variety of 'standards' have been proposed that specify the exact physical and logical format of data. The major reason for this is that the type of networks for which individual standards have been developed differ substantially in what adequately defines a minimal or maximal package of information. For example, a multicomputer network with many peripherals would require sophisticated addressing of devices and suitable error recovery techniques which may require action by devices not necessarily using the network at the time of a network or local communications fault. Just two computers communicating on the other hand, may dispense with the addressing techniques. Clearly, the two systems outlined above represent opposite ends of the communications complexity spectrum but they illustrate that optimal protocols may be entirely different. ' The initial SABRE network is more closely represented by a system of the latter type, and the subsequent protocol used represents its structure. It is envisaged however, that many radar controllers may be interconnected in future with a suitable expansion of the protocol used.

6.1 Overview

The transmission of digital data along the standard PSTN (Public Switched Telephone Network) was the technique adopted for the SABRE communications system. This method is desirable because it allows relatively high speed communications (2400 Baud up to 9600 Baud in some cases), whilst keeping the extra hardware requirment down to a minimum, thus providing a cost effective solution. Within this type of serial communication system there exist two very different methods of data transmission procedures.

The first of these is asynchronous serial data transmission. The form of the digital data that is transmitted is shown in figure 6.1 and constitutes a frame of data for one character. The term asynchronous means literally without synchronisation and no clock information is transmitted separately along with the data. The reception of each character is re-timed from the trailing edge of the START bit and thereafter each bit of data is sampled at a fixed rate which must be the same as the transmission rate. After transmission of each character the line remains idle for up to two bit periods, called the STOP period. This classically was to allow the old mechanical teleprinters to reach a known mechanical position before the reception of the next character.

Usually the state of the line is sampled initially at some multiple of the bit rate; 16, 32 or 64 times. This is done in order to detect the logical 1-0 transition that indicates the presence of the START bit as soon as it occurs. Spikes on the line may be effectively filtered at this stage by checking that

the line remains in the SPACE state for at least one half the bit period after the transition. After a valid START bit has been detected the remaining bits are sampled once, roughly in the centre of their transmission period.

This sampled serial data are shifted into a register that holds the eight data bits after eight shift periods. This data may then be read as a parallel word by the computer. This is known as single buffered mode in which the computer has only the time of the ensuing STOP period to be able to read the data before the next character starts to be assembled. If the computer fails to do this an OVERRUN condition arises to indicate that data has been lost and is usually represented by a flag being set, by the hardware, in the interface status register.

A better method that allows more time is known as a double buffered mode. In this case when a character has been assembled the datum is read by hardware from the serial register as a parallel word, to a second latch, which may then be read by the computer. Thus the next character may be built up in the serial shift register whilst the computer is reading the previous word from the data latch. Nevertheless, an OVERRUN condition can still arise if the computer fails to read the data from the latch during one character transmission time. The double buffered mode provides the software reading the data with more time to process characters than just the STOP period as in the single buffered case.

It is interesting to note that the overhead associated with asynchronous transmission is a function of the number of extra bits required to be transmitted in order to frame the character so that it may be received properly. As figure 6.1 shows, for 8 bit characters, with 2 STOP bits, 1 START bit and 1 PARITY bit, the overhead is 4 bits in a total of 12 bits. This represents a overhead of 33.3 percent for each character transmitted and also therefore represents the same overhead for a message of arbitrary length.

Because the timing information is contained within each frame and is used by the hardware to locate the bits of each character, data may be both transmitted and received simultaneously by the interface, providing that two distinct physical channels exist for transmission and reception. For a computer terminal (a common application for asynchronous links), separate wires are used to connect the transmitter of the computer interface to the receiver of the terminal and vice versa. For a modem communications system only one line exists between the modems at the two sites, yet full duplex type of operation is still possible. It is achieved by using FDM (Frequency Division Multiplexing) techniques, that divide the available frequency bandwidth into two distinct channels. The modem converts the digital signal into FSK (Frequency Shift Keyed) signals which may then be transmitted along the telephone line. Received signals are FSK signals that operate on different frequencies to those that are transmitted by a particular modem. So typically a modem will transmit a binary 1, or MARK, as a 1300 Hz signal and a binary 0, or SPACE, as a 1700 Hz signal. This

modem would then expect to receive a MARK as a 1800 Hz signal and a SPACE as a 2200 Hz signal. The modem with which it is communicating obviously has to transmit on 1800 and 2200 Hz and receive on 1300 and 1700 Hz. Most modems provide operation on both sets of frequencies as a switchable option for greater flexibility.

6.2 Synchronous Data Transmission

This description of asynchronous communications systems has been included to facilitate a more full discussion of its counterpart, namely synchronous communications systems. As was mentioned previously there is a large overhead inherent in asynchronous systems and this becomes intolerable when optimum data rates are to be achieved. Synchronous communications, as the name implies, requires either a separate clock lead from the transmission point to the reception point, in addition to the data lead, or a modem that includes clock information as a part of the modulation process that encodes the data. In typical synchronous modems, the modulation process is a form of PSK (Phase Shift Keying). The clock information is recovered from the sidebands of the received signal and is brought out of the modem on a separate lead that indicates, to the data terminal equipment (DTE), the appropriate instant to sample the data on the received data lead. The synchronous data format is shown in figure 6.1.

Since the START and STOP bits are not required, the overhead per character is eliminated completely, there are however some overheads involved in transmitting a complete message and receiving it correctly.

For a synchronous receiver to know what constitutes the least significant bit of a character and thus be able to assemble characters correctly, and not bit shifted, a synchronisation sequence must initially be transmitted. This sequence is composed of pre-defined SYNC characters which are not a regular

bit pattern. This is done to avoid regular interference patterns being construed erroneously, by the hardware, as synchronisation characters. For example 10101010 would be unsuitable as a SYNC character, the usual bit pattern chosen is 00010110 which is 26 octal. When a synchronous receiver is initially turned on, prior to receiving data, it is in the SYNC search mode. In this hardware mode bits are shifted into the receiver and the contents of the shift register compared with the pre-defined SYNC character. When a match is found the next 8 bits are clocked in and also compared to the SYNC character. If they are not the same then the SYNC search procedure is repeated. If they are the same, the hardware considers itself to be synchronised and will indicate to the computer that a character is available after every subsequent 8 bits have been clocked in. The requirement that 2 SYNC characters to be received before synchronisation is complete gives the hardware greater protection against premature synchronisation by line transients or crosstalk effects.

Because bits arrive at a steady, predictable rate in synchronous systems, modems that use phase modulation and other techniques dependent on constant data flow may be utilised. For example, if a signal is transmitted with four different phases, say 45, 135, 225, and 315 degrees then 2 bits can be represented by them, namely 00, 01, 10 and 11 respectively. Thus transmission of the phase changes at a rate of 1200 per second results in an effective bit transmission of 2400 bits/second. These four phases are a typical way of encoding for 2400 baud modems, whereas 16 different phases will normally be used for

9600 baud modems.

In order to be able to transmit a constant flow of data the transfer is not, in general, truly full duplex, as in asynchronous systems. In fact data is transferred in what is known as a simplex mode where station A first transmits to station B, then there is a line turnaround phase where A prepares to receive and B prepares to transmit. The rules for achieving this are controlled by the software, and are defined within the system protocol, which avoids both A and B either both simultaneously transmitting or receiving.

6.3 Protocols

The protocol in a communmcations system is basically a set of rules carried out by the software to govern the operation of the system. The rules are designed to control operating problems in the following areas:-

(1).FRAMING The determination of synchronisation and the correct location of the initial bit of a character.

(2).ERROR CONTROL The detection of errors that have occurred during transmission of the data by means of some form of redundancy check, for example parity checks.

(3).PACKET CONTROL The characters that constitute a message or packet of information need to be controlled so that the data are compatible with the receiving station.

(4).TRANSPARENCY The characters sequences that enable special control characters to be differentiated from raw data.

(5).LINE CONTROL The determination of line procedures to avoid both stations transmitting or receiving simultaneously.

(6).TIMEOUT CONTROL The solution of the problem of messages that for some reason suddenly cease or if one packet of information requiring a response goes astray.

(7).STARTUP CONTROL The ability to start the transmission of data by putting the communications channel into a known state after it has been idle for some indeterminate period.

These seven areas give a general scope as to the requirements of the protocol and are not intended to be definitive. A standard exists (International Standards Organisation 7 level model) which defines the requirements of a protocol, but it is not discussed here since it defines a very general case, the above being more applicable to the case of SABRE.

There exist two basic types of protocol, character oriented and bit oriented. A character oriented protocol such as BISYNC uses special characters to delineate various fields of a message and to provide control functions. Bit oriented protocols on the other hand, work on a bit by bit basis and may detect control sequences at any time. Generally speaking bit oriented protocols are more efficient since they require only a minimal sequence of bits to indicate a particular function, wheras character oriented protocols may require several characters, each composed of 8

bits, to implement a function. The overhead is therefore less, in bit oriented systems, and accounts for its use in high speed multi-computer local area networks. SABRE uses a character oriented protocol since the hardware was designed to implement this type of protocol most efficiently.

6.4 SABRE Protocol

The protocol designed for the SABRE system reflects the very particular nature of the communications hardware, and in particular that only two computers were required to communicate simultaneously. A new protocol was needed to satisfy the restriction that the software must be memory resident at all times. This was necessary to be able to implement the communications software, not only at the SABRE sites, but also at the existing STARE sites, without any hardware modification of the existing computer systems. There have been a number of published papers on how a protocol may be designed formally, but the methods are generally very long and do not guarantee the most efficient solution (Conrad,1980; Bochmann and Sunshine,1980; Danthine, 1980; Bochmann,1980). The SABRE protocol was designed using a finite state approach to achieve formal validation as outlined by Pitrozafiropulo et al (1980).

The initial requirement was the specification of the data format since the packet of information is the most basic element of the whole system. This packet needed to contain the fundamental elements of synchronisation, control information and data. In fact two packet formats have been defined and are shown

in figure 6.2.

The control sequence contains just control information, and the data sequence may contain both. The transmit program interprets the data presented to it in the following manner. When a binary character of 8 bits has the same bit pattern as the DLE (Data Link Escape) sequence then an extra DLE character is transmitted, thus forming a DLE DLE sequence. The receiver software on detecting this dual sequence eliminates one DLE character and carries on, thus reproducing the original sequence. When a control sequence is required just one DLE character is transmitted followed by the control character. Such valid characters are STX (Start Of Text) and ETX (End Of Text). A full list of the valid characters are given in table 6.1.

STX	Start Of Text
ETX	End Of Text
ENQ	Enquiry
ACK	Acknowledge
NAK	Negative Acknowledge
SYN	Synchronise
EOB	End Of Block

Table 6.1

It should be noted from figure 6.2 that there are four SYNC characters transmitted to increase the probability that the receiver hardware gets at least two SYNC characters sequentially to achieve correct synchronisation. It is required that the receiver be synchronised for each data and control block. There is a 5 word overhead as shown in figure 6.2 for each data block

and since the actual data transmitted is usually 100, 16 bit words, this represents a 500/105 = 4.6 percent overhead, which is considerably less than the 30 percent overhead typical in asynchronous systems. Clearly as the length of the data block increases the overhead percentage decreases. The problem here is that as the length of the block increases the susceptability to errors in transmission, thus requiring retransmission of the whole block, also increases. Retransmission therefore puts the effective overhead at more than 50 percent for that particular block. The block length was chosen as a trade off between high efficiency and low susceptability to errors.

The finite state diagram shown in figure 6.3 and figure 6.4 shows the complete protocol for the SABRE system and includes the possibility of error recovery from data in error and complete loss of data. The stable states of the system are represented by the circled numbers shown in figure 6.3 and 6.4. In these states the system is either preparing for transmission or waiting for reception. The interconnecting arrows between states show the type of response that forces the system from one state to another. The actual response is shown alongside the arrows as either a positive number, indicating a received block, or a negative number, indicating a transmitted block. The meaning of these numbers are given in table 6.2.

0 = Data block without errors

1 = Data block with errors

2 = Acknowledge block

3 = Negative Acknowledge block

4 = Enquiry block

5 = End of Block

6 = Unrecognised control sequence

7 = Receiver timeout

N = Any status not otherwise explicitly defined.

Table 6.2

As can be seen from figure 6.4 three stages are identified and these are; (1) call connection and identification; (2) data transfer and integrity and finally, (3) call disconnection and line clearing. The first stage, call connection, requires the initiating station to manually or automatically dial the

answering station and thus establish the physical connection. This stage also specifies the correct transfer of a control block from the initiating station to the answering station and that this block should be acknowledged by the answering station. At this time both computers are in a mutually acceptable state. The second stage is then entered. This is the transfer of a file from one computer to the other and would be composed of a number of data blocks. The basic sequence of events is the passage of one block which is then positively (ACK) or negatively (NAK) acknowledged by the receiving computer. A negative acknowledge causes retransmission of the block in error, whereas a positive acknowledge causes the next block to be transmitted. Clearly not only data blocks may be received in error but also control blocks may be corrupted. As the finite state diagram shows there are a number of error recovery sequences that may be initiated.

6.5 Error Recovery

A burst error is defined as a potentially continuous corruption of data over a period of time. This type of induced error is common in telephone network communication and may be caused by sporadic electrical interference, or exchange switching interference. Since the major requirement of a communication system is the error free transmission of data between end users, such errors as may occur need to be both detected and corrected. To this end some form of redundancy needs to be added to the data.

6.6 Parity Codes

One of the most common methods, but least effective for burst errors, of adding redundancy is the parity check. For a binary word of N bits the parity of 1 bit is added making an information length of N+1 bits. This bit is added to make the number of 1's in the information sequence either an odd (odd parity) or an even (even parity) number. For the message sequence of 10101001 the parity bit, for even parity, would be a 0, but for odd parity would be a 1. By counting the 1's in a received information sequence one bit corruptions may be detected. However multi bit errors may go completely undetected. Consider the message 11011011 transmitted with even parity as 110110110 and then corrupted to 000110110. The receiver would not see any error here. Thus parity checks are unsuitable for even numbers of multi bit errors. One solution to this problem is to add two parity checks both vertically and horizontally across a message block. Consider the message block :-

01101011 (1)
11010010 (0)
11011110 (0)
11101101 (0)
00010101 (1)
(10011111)

The parity checks that would be added are shown in brackets. The message would be transmitted from the top left bit horizontally across the first word, and then left to right across the subsequent words. The parity on the right hand side checks

the horizontal binary sequences, whilst the final word transmitted checks the vertical binary sequences.

This more elaborate method is a considerable improvement over the single parity check, but still suffers from the burst error problems. Its great virtue is that it is easy to implement in either hardware or software.

6.7 Cyclic Redundancy Checks

The CRC (Cyclic Redundancy Check) is the method of error detection employed in most communication systems. It is a result of a trade off between good error detection performance and ease of implementation, usually in hardware. No error correction is usually implemented in communication systems as the protocol used provides adequate error recovery and thus correction.

In all error detection systems a number of redundant check bits are appended to the message bits to constitute the full information sequence that is transmitted. Consider a message polynomial G(X) and a generator polynomial P(X), where X^N represents a 1 as the Nth bit of a binary sequence. Therefore a polynomial X^4+X^2+1 is 10101 as a binary sequence. The object is to construct a code message polynomial F(X), such that it is evenly divisible by P(X). This may be achieved as follows:-

(1).If there are K message bits and N information bits then the number of check bits required is N-K. Initially the N-K check bits are preset to zero and appended to the K

message bits. This gives an initial sequence of $G(X).X^{(N-K)}$ which make the N information bits.

(2).The N information bits are then divided by the generator polynomial P(X).

(3).The quotient is disregarded but the remainder C(X) is added to the initial product $G(X).X^{(N-K)}$ thus replacing the N-K zero check bits with the N-K remainder bits. The final coded message may be represented as $G(X).X^{(N-K)} + C(X)$.

Consider the message of 110011 = G(X)

and a polynomial 11001 = P(X)

The number of check bits added is always one less than in the generator polynomial.

Then $G(X).X^{(N-K)}$ = 1100110000

Note the 4 zero's as the least significant bits.

The division in stage two is done without any carry or borrow and may be programmed in the following manner. The actual division is performed by taking the successive modulo 2 exclusive OR of the product and the generator polynomial P(X)

Thus

```
                100001
          11001 1100110000
                11001
                    10000
                    11001
                     1001
```

The remainder C(X)=1001 which when added to the initial product yields an information sequence of 1100111001.

This message may then be transmitted and at the receiving station the error detection is performed. The received message is divided by the generator polynomial and if no errors have occurred then the division remainder would be zero. For the above example the division is shown below.

```
                100001
          11001 1100111001
                11001
                    11001
                    11001
                    00000
```

Typically a 16 bit check sequence would be used and therefore a 17 bit generator polynomial is required. A CCITT standard polynomial is given below and is indeed the one used in the SABRE system.

$$X^{16}+X^{12}+X^{5}+1 = 10001000000100001$$

This provides error detection of bursts up to 16 bits in length, additionaly more than 99 percent of error bursts greater than 12 bits may be detected. Longer generator polynomials provide better error detection capability but the length given above combines an error detection performance, that is usually more than adequate, with ease and speed of performing the division on a 16 bit machine. Although the generator polynomial is 17 bits, in practice this provides no problem since the division is performed when a binary 1 is the most significant bit of the information sequence, which may then be shifted into the carry bit of the accumulator. The exclusive OR of the most significant bit of the generator polynomial and the carry bit is always a 0 by definition. Thus the problem reduces to merely taking the exclusive OR of the remaining 16 information bits and the remaining 16 generator polynomial bits. The sequence that is divided by the generator polynomial is the total number of characters that are to be transmitted as a continuous binary stream. Therefore if say 100 characters each of 8 bits are to be transmitted then the 800 plus 16 initially zero bits are treated as one binary sequence to be divided by the generator polynomial.

Because the implementation of this type of code reduces to two basic types of operation, namely shifting and exclusive OR, it lends itself very well to a hardware implementation, however in the initial SABRE system the CRC was calculated in software for initial testing.

6.8 System Software

The software was written to control the Data General ULM5 (Universal Line Multiplexor) modem interface. This board contains just one synchronous line and has full modem control features. The ULM5 allows the transmitter and receiver to be turned on and off under program control and allows the software to preset the SYNC characters and DLE characters that the hardware is required to recognise. The line characteristics may also be programmed and this dictates the number of bits per character that are to be transmitted and the type of parity check, if any, that is to be appended to the character. Since the maximum character length, including parity, was 8 bits, 8 bit characters were specified but with no parity. This was done so that a 16 bit word could easily be divided into two, 8 bit characters for transmission. Otherwise if say 7 bit characters plus even parity was chosen the first two characters would occupy the first word to be transmitted, but the next character would occupy the remaining 2 bits of that word and the first 5 bits of the next word. If ASCII only was to be transmitted this difficulty would not be encountered as the ASCII character set is defined to be only seven bits, with the remaining bit normally preset to zero. Thus two characters would occupy one word completely. However to transmit binary object codes there is no redundancy in the 16 bit words, as there is for ASCII characters, and so to avoid the time consuming splitting up of data descibed above, 8 bit characters were chosen and the hardware parity option on the ULM5 disabled. This is another reason why the software CRC check was chosen to be appended to the data.

The software structure is shown in figure 6.5. It should be noted that the Fortran routines were included as an easy means to change the protocol during the testing stage but were programmed in assembler for the computer sited at Wick. The assembler subroutines were assembled independantly of the Fortran but each contains a suitable Fortran interface to enable the passage of parameters. The Fortran and assembler were combined properly at load time to construct the complete object module.

SETUP calls an RDOS system routine IDEF, that includes an entry in the interrupt despatch table for device 34, namely the ULM5. This is necessary since a system running under RDOS controls all system states including DMA (Direct Memory Access) and Interrupts. An interrupt from device 34 is defined by the routine SETUP to be execution of subroutine ISERV. The routine ISERV merely reads the ULM5 and sets flags MFLG, RFLG, or TFLG depending on whether the interrupt comes from the modem, receiver or transmitter section of the ULM5 respectively. If it is the transmitter section that is requiring service a character from a common location labelled TDAT is loaded into the transmitter. If it is the receiver requiring service then the character is read from the ULM5 receive buffer and then stored in the common location RDAT. The modem status is placed in the common location STAT.

By using these flags the transmit and receive routines (TXMT and RXMT respectively) are able to determine when more data are required or when more data are available. Thus these routines wait until the flag is set before taking appropriate action.

One of the parameters passed to the TXMT routine defines the block type that is to be transmitted. If a control block is to be transmitted then the extra DLE characters are not inserted since it is in this case desired to transmit control sequences. If a data block is to be transmitted then the first 6 characters are transmitted without extra DLE characters, since for every block the control sequence DLE.STX must be recognised by the receiver program. Then the data are transmitted with the necessary extra DLE characters. Finally the one word CRC and the DLE.ETX control sequence are transmitted without the extra DLE characters. The data block length is passed as a common parameter from the routine READL and is labelled DATL. This allows dynamic changing of the transmission data block length to optimise the throughput in relation to the observed errors.

The RXMT routine receives characters but does not begin to store them until it has received a DLE.STX sequence. Thereafter it receives characters and stores them, keeping a running status on the length of the block and control sequences received. If a data block is received then a CRC is computed over the data and the remainder is compared to zero. Reception of a DLE.ETX sequence indicates to the receiver that the block is ended at which time the receiver hardware is turned off by the software and the data processed. There are two main error conditions that are trapped by the routine RXMT. Loss of line data completely will cause RXMT to timeout and pass an appropriate error code back to the Fortran protocol controller. Secondly, if the end sequence DLE.ETX is corrupted then the receiver will continue to

clock in characters at a steady rate. It is probable in this case that the transmitter at the initiating station will be turned off and the line will thus be in the MARK state. This means that the characters received will be 377 (octal). However the routine RXMT will only allow a maximum block length, if this is exceeded the routine will pass an error code to the Fortran that would probably ask for retransmission of the block.

Control of the modem is an essential part of the data communications system since it is this that is responsible for the physical transmission of the data as well as the connect and disconnect procedures. Several lines are available from the modem to the data terminal equipment which is, in this case, a computer. These lines and their control are reviewed below.

REQUEST TO SEND (RTS) is provided to turn the modem transmitter on and off. RTS=1 turns the transmitter on and data may then be sent to the modem along the TXDATA line. When RTS=0 the modem transmitter is turned off but the modem is available to receive data which it then passes along the RXDATA line.

DATA SET READY (DSR) is a signal from the modem to the computer to indicate that the modem is operational and not in test mode.

CLEAR TO SEND (CTS) this is a signal provided by the modem to indicate that it is ready to accept data for transmission. This line may be ignored, and in this application the CTS input to the ULM5 is hardware asserted. It may be normally assumed that the modem is always ready to accept data for transmission.

The above three lines represent the basic handshaking between the interface and the modem, but there are another three that are required to provide the necessary status for switched network operation of modems.

DATA TERMINAL READY (DTR) this line is asserted by the computer to enable the connection of a call from an incoming station. DTR may be left asserted to enable incoming calls to be answered automatically. However when this line is no longer asserted the modem disconnects the call.

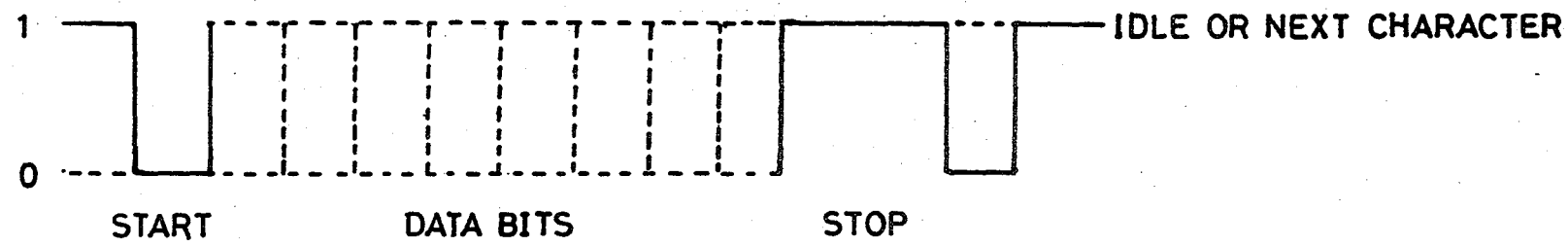
RING INDICATOR (RING) as the name implies this line indicates when the modem has been called by a secondary station. Assertion of this line causes the modem status register in the ULM5 to be updated and thus to issue an interrupt to the computer. After the computer has determined that it was the ring status that caused the interrupt the usual course of events would then be to assert DTR. Alternatively if DTR is permenantly asserted then the call would be connected immediately.

DATA CHANNEL RECEIVED SIGNAL DETECTOR this line must not be confused with anything to do with the DMA lines on the Nova also referred to as DATA CHANNEL. In fact this signal is used to differentiate between incoming data and incoming voice calls. One of the problems which faces a data communications system is that of misdirected voice telephone calls. Assume that a person making an ordinary voice telephone call accidentally reaches a modem. Ring indicator is asserted to the modem interface and the computer asserts DTR thus connecting the call. At this time the software starts a timing sequence to check for assertion of the

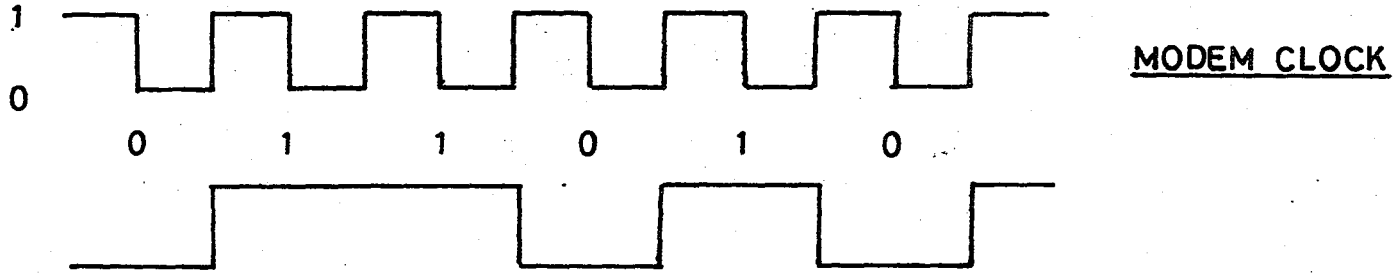
carrier from the calling modem, as would be indicated by assertion of the data channel received signal detector. Since in this case the caller is a person, no such assertion will occur and the timer will expire. This would normally then disconnect the call.

6.9 Conclusion

The ability to communicate with a remote expertiment has many advantages that have been discussed in the previous chapters. What has been described here is a small, high speed system, consisting of the minimum hardware and software required to implement a reliable communications link. It has been shown that a communications link may be established to a small computer system and that it is possible to provide continuous data flow from the experiment in 'real time' and previously recorded data. This latter point is important since for reasons of economy the link is only established for approximately 6 hours a day. The data taken by the radar during the time the link is not established is written onto magnetic tape as has been previously described. When the link is established the 'real time' data are only present at the end of each integration period, typically every 20 seconds. This data only takes about 1 second to transmit so there is sufficient time to transmit previously recorded data, before the next 'real time' data are available. In this way 'real time' data are interleaved at the analysis site with previous data. Since all data has a time associated with it, the sorting is a relatively simple task.

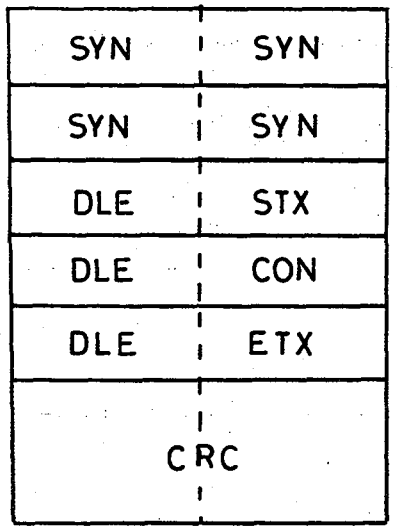


ASYNCHRONOUS CHARACTER

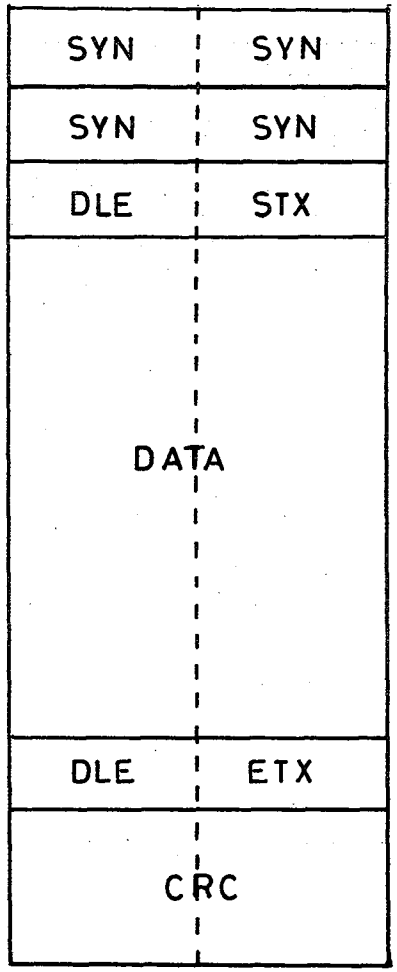


SYNCHRONOUS DATA FORMAT

FIGURE 6.1

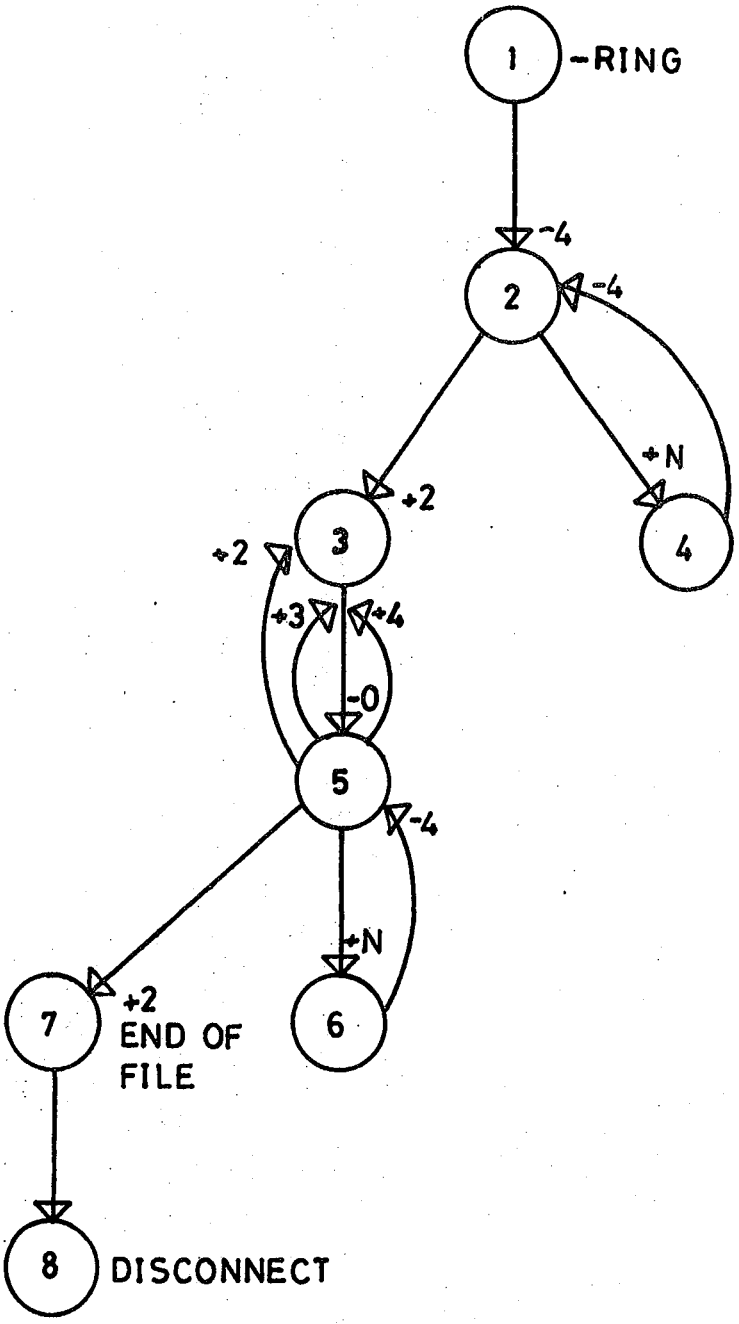


CONTROL PACKET



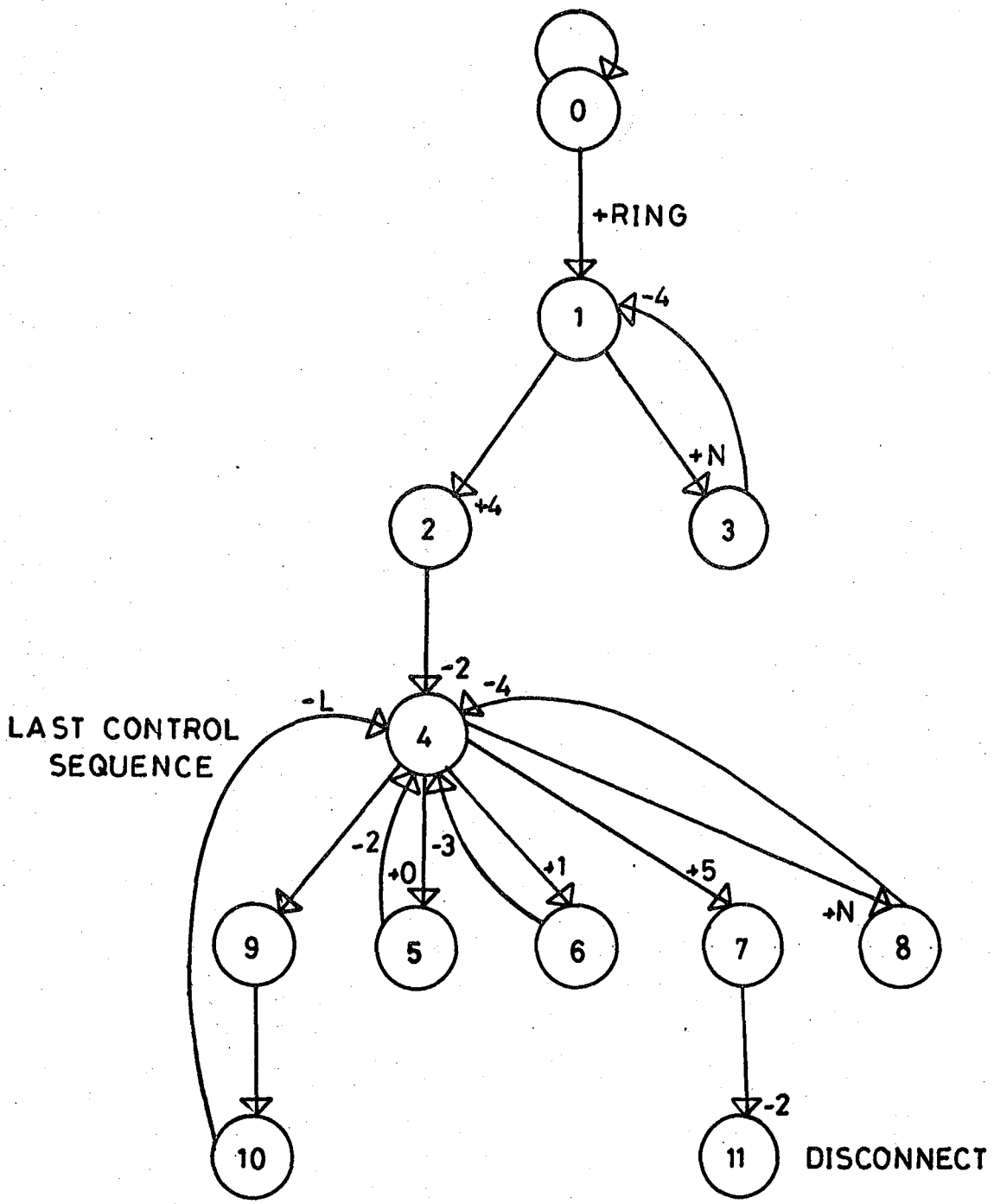
DATA PACKET

FIGURE 6.2



FINITE STATE DIAGRAM OF TRANSMITTER

FIGURE 6.3



FINITE STATE DIAGRAM OF RECEIVER

FIGURE 6.4

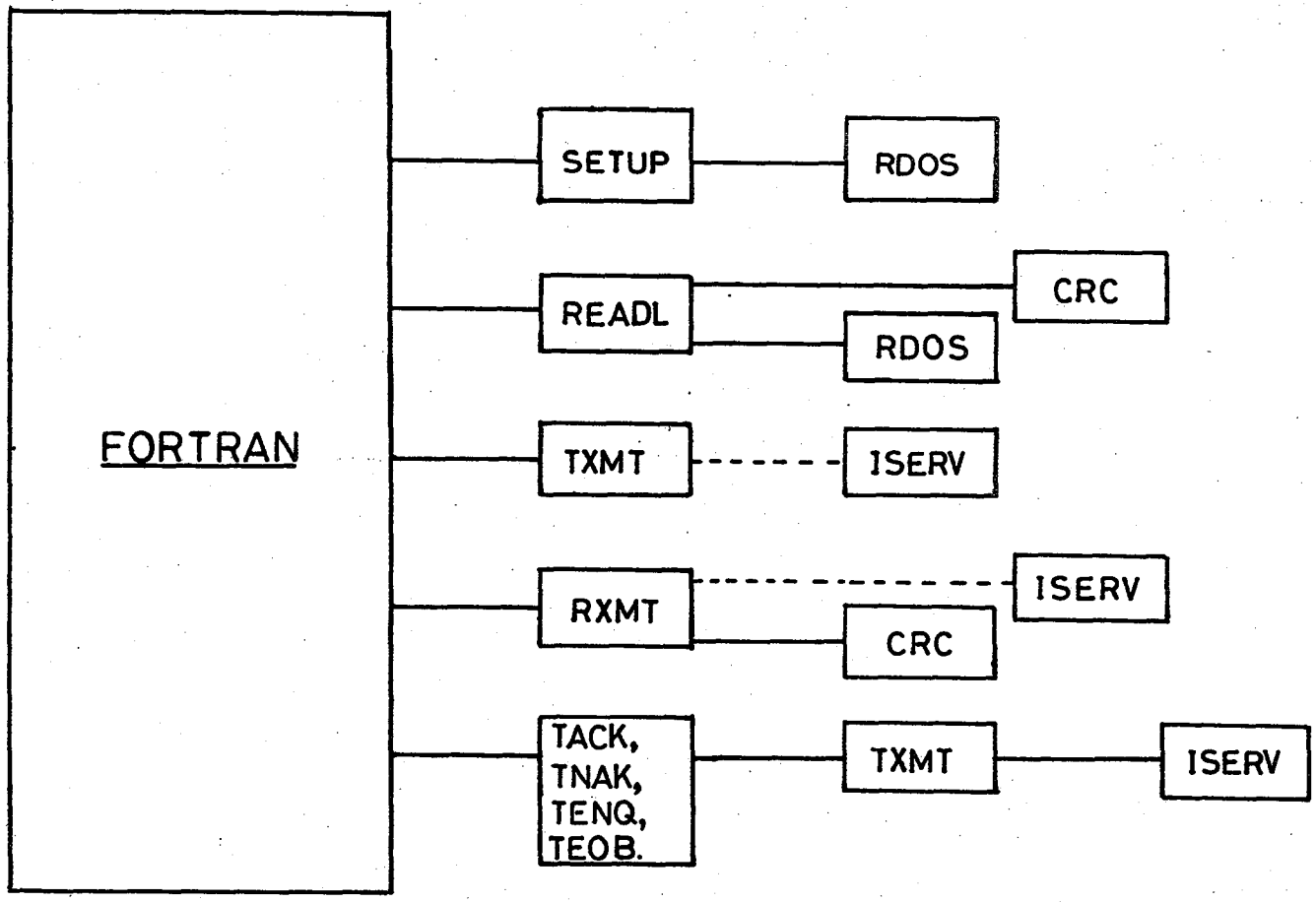

FORTRAN
SETUP
RDOS
READL
CRC
RDOS
TXMT
ISERV
RXMT
ISERV
CRC
TACK,
TNAK,
TENQ,
TEOB.
TXMT
ISERV

FIGURE 6.5

Chapter 7

Computer Interfaces, Design and Programming

7.0 Introduction

This section describes in detail the operation of the hardware interfaces that were designed for the SABRE system. The previous chapters have discussed the motivation for the particular design criterion, and so this section concentrates entirely on the design of the electronics. An attempt has been made in the description to unify the circuit design with the software required to run it. The interrelation of these two considerations is important to ensure efficient and flexible hardware control. A discussion of the DMA and interrupt structure of the Nova computers serves to introduce the description of the device interfaces.

7.1 DMA Transfers

The direct memory access transfers on the Nova computer are called Data Channel Transfers. Two lines of the I/O bus are allocated to determine the nature of the transfer required and are controlled by the device interface issuing the request. These two lines (DCHM0 and DCHM1) enable four kinds of transfer:-

DCHM0	DCHM1	
0	0	Data Out
0	1	Increment Memory
1	0	Data In
1	1	Add to Memory

The last option enables a word to be added to the current contents of the referenced memory location rather than just replacing it. This mode has application in signal averaging in which samples of a waveform taken on successive cycles at the same point need to be accumulated. As a means of reducing the number of lines taken to a device interface (which may be remote from the CPU chassis) the address of the DCH transfer is passed along the same bus as the data. To enable correct timing of these operations the CPU asserts the data channel accept (DCHA) control line to indicate that an address is required and data channel I/O lines are asserted, (DCHI and DCHO respectively) when data is to be input to, or output from, the bus. To allow for maximum settling time on the data lines, the CPU gates the address and data into memory at the end of their respective control signals.

Several data channel devices may be installed in the chassis (or external to it), the priority being determined by their physical location on the bus. In order to decide which device has priority, the processor sends out a data channel priority signal (DCHP). This signal is taken through logic in all devices on the bus. When a device issues a request, which it does after checking to make sure that its DCHP input (DCHPIN) signal is high, it then sends a low DCHPOUT to the next device. All other devices must consult this signal prior to issuing a request to the CPU. The device nearest the CPU will gain priority over succeeding devices on the bus through which the DCHP signal will pass unhindered.

All data channel requests are synchronised to the CPU activity, the flags which are required to achieve this are shown in figure 7.1. When a data channel transfer is required, the issuing logic sets the data channel synchronising flip flop. This signal is used as an input to the next flip flop (device data channel request), which is set on the next leading edge of the CPU timing signal RQENB. The output from this flip flop is buffered to drive the open collector "WIRED OR" data channel request line, DCHR. However the device interface must not reset the data channel request until it has been acknowledged. This is ascertained by DCHPIN being high and the request flip flop output also being high.

Each device on the bus is responsible for terminating its data channel transfers. The above sequence of accessing the CPU must be initiated for each word transferred. To achieve a multi-word transfer two registers are required, one to hold the current memory address of the transfer and one to count the number of words transferred. The transfer is terminated when the word count register overflows since the 2's complement of the number of words to be transferred is usually stored. For each word transferred the address register and the word count register are incremented. The overflow from the word count register can then be used to set the local device DONE flip flop, which can then be tested by the software.

7.2 Programmed I/O

The NOVA line of computers have four general purpose 16 bit registers and data can be transmitted between any one of them and an external device. The programmed I/O instruction format is shown below. Each device to be programmed can have up to 3 registers into which data can be written to or read from. The register concerned together with the direction of data flow, is indicated by bits 5, 6 and 7 of the instruction format as follows:-

5	6	7	DIRECTION	LINE CONTROLLED
0	0	1	IN	DATIA
0	1	0	OUT	DATOA
0	1	1	IN	DATIB
1	0	0	OUT	DATOB
1	0	1	IN	DATIC
1	1	0	OUT	DATOC

The I/O bus of the NOVA comprises of 50 lines of which there are:-

(1) 16 bi-directional data lines (open collector)

(2) 6 device select lines

(3) 28 uni-directional lines carrying control signals

The signal levels are nominally between 0 and 0.8 Volt for logic 0 and between 2.7 and 5 Volt for logic 1.

Each interface needs a number of single bit flags to control its operation and indicate its state back to the processor. Two of these, the BUSY and DONE flags, are controlled by bits 8 and 9 of the instruction word. Their state is indicated to the processor during each I/O test instruction along the lines SELB and SELD. This allows the CPU to respond to various SKIP instructions that depend on the state of the BUSY and DONE flags. The final six bits of the instruction word indicate the device code to which the instruction is destined. Each interface decodes these six lines to provide a unique signal which indicates to the rest of the interface whether action is to be taken or not. Figure 7.2 shows the device select and BUSY/DONE logic.

7.3 Interrupt Logic

The DONE flag on any interface can be used to initiate an interrupt request. This enables any device to signal a service request when an operation has been completed. To signal an interrupt request the INT REQ flip flop is set on the next leading edge of RQENB after a device has initiated a request. When the processor receives this signal it sends out an interrupt priority signal INTP. This line is connected in a similar manner to DCHP, through all devices in the interrupt chain. Considering two devices in the interrupt chain for simplicity, when neither device is requesting an interrupt, both INT REQ lines are high. The INTP signal passes through the network, starting as a logic high and remaining so at the end. If, however, the first device is requesting an interrupt the output from the first NAND gate is

forced high thus ensuring that all subsequent DEV ACK (device acknowledge) signals remain high. Thus when INTP goes low indicating the processor is ready to respond to the interrupt only one DEV ACK signal will be generated, which is for the highest prioriy device requiring service. This signal, DEV ACK, is then used to gate the INTA (interrupt acknowledge) signal which drives the bus buffers. When INTA is asserted the device code of the device requesting service is placed on the bus. This provides a quick way of determining which device is currently needing attention, and this device code is normally used as part of a hashing algorithm to provide the address of the interrupt service routine.

The effect of a hardware interrupt is to vector the processor via location 1 and the return address is saved in location 0. Thus the sequence of events following an interrupt would typically be :-

1..Hardware interrupt occurs current address saved by the hardware in location 0. Processor executes JMP @1.

2..Interrupt service routine typically executes INTA 3 thus getting the device code in accumulator 3.

3..A table of service addresses for each device may be typically at location 200, where the address for a device of device code N is stored at location 200+N. Executing JMP @200,3 would then execute the appropriate routine.

4..After servicing, the program enables the interrupts (INTEN) and returns to the interrupted program JMP @0

During a service all other interrupts are disabled and must be enabled just before returning to the main program. Devices may be prevented from requesting interrupt service by executing the mask out instruction, MSKO. All devices have a one bit flag which is connected to one bit on the data bus. When MSKO is executed all flags are loaded according to the contents of the accumulator specified. A "1" will disable interrupt from all devices connected to that line. The restriction here is that only 16 groups of interfaces may be independently controlled, but this is usually sufficient.

The use of MSKO can be used to implement a multi-level interrupt structure. A device may request an interrupt, the hardware thus disabling all other interrupts. The interrupt service routine can then issue a MSKO preventing all devices but the ones allowed, to interrupt the interrupt service routine. Issuing an INTEN completes the structure whilst still servicing the first device requesting service. The service routine may also alter location 1 thus changing the primary course of action taken on subsequent interrupt requests.

7.4 NOVA 3 Interfaces

The preceding sections have been included to provide a basis for discussion of the SABRE interfaces and to enable the circuit diagrams to be followed more closely. The primary differences between the NOVA interfaces and MICRONOVA interfaces are mainly due to the more recent development of the MICRONOVA. The MICRONOVA has a large scale integration (LSI) chip (MN603) and a

few support chips to implement most of the basic priority and timing functions described above, these functions have been implemented using the standard small scale integration (SSI) chips on the NOVA.

7.4.1 Pulse Interface

The pulse interface provides the shaped and TTL compatible pulses for the transmitter. The pulse shape is stored in a 256x8 bit memory comprising of 2x2114 static random access memories (RAM'S). A clear (CLR) pulse sets the memory address counter to 0 and this corresponds to the first word of the pulse shape. The clear pulse also selects the multiplexer to allow loading of the RAM. A DOAP instruction enables the 74125 tri-state buffer during the time the DOA control line is low. This line also is also connected to the R/W line of the memory thus selecting WRITE. Data on the bus during this instruction is thus strobed into the memory. The second part of this instruction is to pulse the IOPLSE control line. This is used to increment the address counter. Thus a series of DOAP instructions will load the memory. This may be followed by a NIOC instruction to set the address counter back to 0.

The TTL pulse needs to be synchronised with the shaped pulse as indicated in figure 7.3. This is achieved by detecting the first zero element in the pulse shape memory as the memory is being read. This "zero detect" is used to terminate the TTL pulse and to set the interface DONE flag. This is primarily a single pulse system which commences with a NIOS instruction to set BUSY=1, DONE=0 initially setting the TTL output="1" and

reading through the memory until a zero is detected. Clearly the double pulse sequence could be implemented by the following sequence :-

```
NIOC 44     ;set address counter=0
NIOS 44     ;run interface
SKPBZ 44    ;wait till finished
JMP.-1      ;loop
.
.
.           ; inter-pulse timing
.
.
NIOC 44
NIOS 44
SKPBZ 44
JMP .-1
```

This requires software timing, and since the wait period is typically 300 microseconds it is a waste of processing time. To overcome this, a second counter is included in the pulse interface. This is loaded by a DOB instruction and represents the interpulse delay. It is loaded with the 2's complement of the absolute delay required since the overflow carry is used to indicate that the time period has expired by setting a flip flop. The clock for this interface is derived from a 10 MHz crystal oscillator and a 74LS90 decade counter providing a basic time period of 1 microsecond. When a NIOS instruction is executed, this clock is gated to the address and inter-pulse delay counter.

Thus the inter-pulse delay reflects the time period between the start of one pulse and the start of the next. If the inter-pulse delay is less than the width of one pulse then the restart pulse is ignored since it is generated during the time the select flip flop is already set. If however the time period has not expired by the time the zero detect is active then the DONE flag is prevented from being set but the address counter is still automatically reset to 0. At the end of the inter-pulse time period the restart pulse initiates a second pulse sequence and enables the DONE flag to be set at the next zero detect.

By this method single and double pulses are available whose length is extendible up to 256 microseconds and the TTL pulse is automatically synchronised without further programming. To select a single pulse the inter-pulse gap can be made 1 microsecond by storing -1 as the inter-pulse gap which is very much shorter than the 100 microsecond pulse width normally used.

7.4.2 A-D Conversion Interface

As has been described earlier the receiver provides quadrature outputs which need to be sampled simultaneously to preserve the phase information present at any particular instant in time. Since there are eight such channel pairs to sample, 16 A-D converters could be used to sample all of them simultaneously. This, however, is pointless in a single computer system as the data is read into the computer in essentially a serial fashion, one, 16 bit word at a time. A much more practical and cost effective solution is to use the minimum allowable A-D converters with a channel multiplexer. Thus by using 2, 8 bit convertors the quadrature outputs for a particular channel can be converted simultaneously and read into the computer as 1, 16 bit word.

Another requirement of the A-D conversion is that all 8 quadrature channels should be converted within the basic resolvable time period of one transmitter pulse width, typically 100 microseconds. To meet the above criterion a high speed data acquisition interface was used that also had channel multiplexing capability.

Two counters are required to implement the data channel address and word count registers. These, and the data channel timing logic, have been considered in some detail earlier and so will not be discussed further here. The other two counters used are for the pre-sample wait and inter-block delay. The pre-sample wait is used as a delay until the first sample is converted, which essentially sets the range of the backscatter

signals. The inter-block delay corresponds to the time between the conversion of channel 1 at range N, and the conversion of channel 1 at range N+1. This delay is set to be the same as the width of the pulses used since this corresponds to the maximum range resolution. The 8 quadrature channels are converted within 50 micoseconds, but the next following sample set is not taken until the start of the next sample period which is accurately timed to within one clock period (1 microsecond) from the start of the last sample period.

The two end of conversion signals (EOC) are combined to provide a composite signal which follows the slowest converter and this provides a data channel request to the computer. Once this value has been read by the computer (DCHI asserted) another conversion is initiated, thus conversion is not overlapped with reading since the converter is very much faster than the computer. The word count register sets the DONE flag when the number of data words required have been transferred and the word count register overflows. Figure 7.4 shows the circuit diagram of the A-D interface.

7.4.3 Real Time Clock Interface

The real time clock has been built around a large scale integration (LSI) clock chip. This chip is a low threshold metal gate CMOS circuit that functions as a real time clock and calender. Time keeping is maintained down to 2.2 volts to allow low-power standby battery operation. The battery backup is provided by a ni-cad 3.3 volt cell that is trickle charged during normal operation. The reference frequency is crystal controlled from a 32.678 kHz oscillator.

The clock chip is designed to operate in a memory mapped computer environment, so that the internal clock registers may be treated as sequential memory locations. To achieve satisfactory operation in an unmapped system, such as the Nova, a considerable ammount of external logic is required. The DOC signal is used to indicate the direction of transfer required, that is, either writing to, or reading from, the clock registers. A "1" in the accumulator, when issuing a DOC instruction, sets the read mode, whereas a "0" sets the write mode. A DOA instruction latches the address specified by the contents of the associated accumulator. An I/O pulse IOPLSE, is extended by the use a 74LS121 monostable. This is because the IOPLSE signal is multiplexed to either the read or write strobe line on the chip (NRDS,NWDS respectivly), and must be at least 0.5 microseconds, the usual length of IOPLSE being 0.1 microseconds. The trailing edge of IOPLSE is used to set the DONE flag indicating to the processor that the data has been written, or read, by the clock. In the latter case the data is placed in an external latch. The processor can then read the

data with a DIB instruction. To write to the clock chip a DOA instruction is used to latch the address of the appropriate register. A DOBP instruction latches the data into an external 4 bit latch and subsequently provides a write pulse in the form of an extended IOPLSE.

7.4.4 Graphics Interface

The graphics interface is constructed around two 8 bit digital to analogue converters. These provide an analogue signal that is a varying current, controlled by digital input. This current is converted to a 0-10 volt signal using a high bandwidth op-amp. This analogue voltage is suitable for driving a storage tube for displaying the data. By using one converter for the X channel and one for the Y channel, two dimensional plots may be obtained. The beam may also be turned off by a transistor switch which is connected to the output of one of the software controlled registers. Similaly the screen may be erased by writing one bit (bit 0) to the register holding the Y co-ordinate.

7.5 Micronova Interfaces - Introduction

The Micronova interfaces have been implemented using the standard "General Purpose Interface Boards" (GPIB). These boards already contain the logic required to implement the correct data channel timing and associated logic for device select, programmed I/O and Busy/Done. They provide a wire wrap area on which the following interfaces have been constructed.

7.5.1 Interprocessor Interface

One of the major requirements of the SABRE computer system is the ability to change the radar parameters without interrupting the operation of the Nova 3. This can clearly be done by using the data channel facility, which requires no software intervention by the recipient computer. The interface was designed to meet this basic need and any other future requirement. Any area of the Nova/Micronova memory can be mapped to any area of the Micronova/Nova memory and once the transfer has been initiated by the Micronova no other intervention is required by either computer. It should be noted at this stage that the Nova cannot initiate interprocessor data channel transfers (IDT's).

The data channel speeds are different between the two computers, that is, the time between initiation and completion of a one word transfer. This time is typically 2 microseconds for the Micronova and 1 microsecond for the Nova. Since data is only present when DCHO is asserted, and data must be available when DCHI is asserted, a one word buffer is required to ensure that

data is latched during DCHO and is thus available during DCHI. The direction of transfer required (Nova-Micronova, Micronova-Nova) is stored in a D-bistable. This is used to select a quad 2-1 line multiplexer which is responsible for gating the subsequent data channel requests. Thus initially the Micronova start pulse STRT will be used to request data channel service from the computer providing data. DCHO from that computer is then used to issue a data channel request to the receiving computer and to latch the data present. DCHI from the receiving computer reads the data from the latch and issues another data channel request to the computer providing data, and the sequence continues.

Whatever the direction of transfer, it is the Micronova word count register that is used to terminate the transaction, when this register overflows the control line WCEZ is asserted. At this time, however, the transaction may not be complete, but only in its final stage of issuing a request to the Nova or Micronova. WCEZ is used to set a flip flop and that is gated with the DCHI of the receiving computer. Thus the Done flag is set only after the receiving computer has stored its final word of data.

The DOA instruction is used to store the Nova starting address for the transfer in a 16 bit synchronous counter. Bit 0 indicates the direction of transfer, 1=Nova-Micronova and 0=Micronova-Nova. DOB is used to store the Micronova starting address, and DOC is used to store the 2's complement of the number of words to be transferred.

The Nova address counter is incremented after each Micronova data channel request and thus when a Micronova-Nova transfer is required the address counter holds A-1 where A is the Nova starting address. When a Nova-Micronova tranfer is required the address counter simply holds A.

7.5.2 Voltage Monitor Interface

The Micronova can monitor up to 32 analogue channels which represent critical voltages and currents in the transmitter and receiver. The interface to do this has been built around two single chip data acquisition systems. These are monolithic CMOS devices with an 8 bit analogue to digital converter, a 16 channel multiplexer and the associated control logic. The chip does not contain a sample and hold gate and one has not been installed, although provision to be able to do this has been made on the chip. The voltages that are being measured should remain reasonably constant, so a sample and hold gate was not necessary. Regardless of whether a sample and hold gate is used, a transient during conversion could lead to an erroneous measurement, however the software effectively "filters" this out by ensuring that a potential error condition is persistant before taking appropriate action.

The channel address to be converted is latched into the register on board the chip with a DOB instruction. Having achieved this a start pulse initiates the beginning of conversion on board the chip. This can be done with a single instruction DOBS. The combined end-of-conversion signal is used to set the DONE flag thus ensuring that DONE is not set until the slowest converter has finished. The DONE flag is tested in the usual way, with a SKP and JMP.-1 sequence to wait until conversion is complete. The converted value is then read using a DIB instruction. The tri-state control (TSC) from the chip is always active since the output lines are buffered through open collector

drivers to drive the processor bus. The clock for the chip is divided from the 10 MHz processor clock to 1MHz. This gives an average conversion time of approximately 100 microseconds per channel.

7.5.3 Bubble Memory Interface

The magnetic bubble memory (MBM) device used in the SABRE system is the 1 Meagabit 7110 chip made by Intel (Intel 1980, Intel 1981a,b). This togeather with the 7220 LSI controller, the 7230 current pulse generator, 7242 formatter/sense amplifier, the 7250 coil pre-driver and the 7254 VMOS drive transistors form the complete system. The 7110 is essentially a serial-parallel-serial shift register with the data being organised in 2048 binary pages, each holding 512 bits. The data is subdivided again into two channels of 256 bits or 64 bytes. There are in fact 128 data storage loops per channel divided into two sections of 64 data loops each. A page address is selected and the appropriate page shifted to the correct position for a read or write operation. The chip is made with a number of redundant storage loops. During manufacture a proportion of the total number of loops will statistically be faulty, thus to increase yield, the chips are made initially with greater than the 1 Mbit capacity. To isolate and identify the faulty loops, a boot loop is installed on the chip. This loop can be read by the formatter sense amplifier (FSA) and bad loops automatically masked out by the hardware.

Data may be transferred in one of three distinct modes; polled I/O which is the slowest of the three; interrupt driven I/O and direct memory access. In the SABRE implementation only the polled I/O method was implemented. Although this is the lowest performance solution it is also the most easily controlled and monitored.

7.5.4 MBM Hardware Operation

The operation of the MBM is controlled by writing to the internal registers of the 7220. When the address line A0 is high the following registers may be accessed:-

Data line 4=1 COMMAND REGISTER

Data line 4=0 REGISTER ADDRESS COUNTER

To access the other registers the address must first be written into the auto incrementing register address counter, then with A0 low the next data written to the 7220 would be written to the register specified. Data is written by strobing the WR line low and data read by strobing the RD line low. It has been found that the A0 line must go low before the RD or WR lines are strobed, so the A0 line is controlled separately by the software by using the NIOS and NIOC instructions and having the A0 line connected to a set/clear flip flop. Once A0 has been set by the software WR and RD may be controlled independently. These strobes are extended from the DOA and DIA lines by the use of monostables. A one word bi-directional buffer has been placed between the bubble memory and the CPU to extend the time that data is available and thus ensure sufficient time for both the CPU and MBM to receive data correctly.

Data to be written to the MBM is first written with a DOA instruction into a latch. This data is latched on the leading edge and is available to the MBM after this time. The extended pulse is simultaneously passed to the WR line and data is subsequently written to the MBM on the trailing edge of the

extended pulse.

Reading data from the MBM is essentially a two part process. A DIA instruction is extended and used to pulse the RD line. Data is written on the trailing edge of this pulse into a latch. Data is subsequntly read from the latch into the CPU with a DIC instruction. The circuit diagram of the MBM interface is given in figure 7.9.

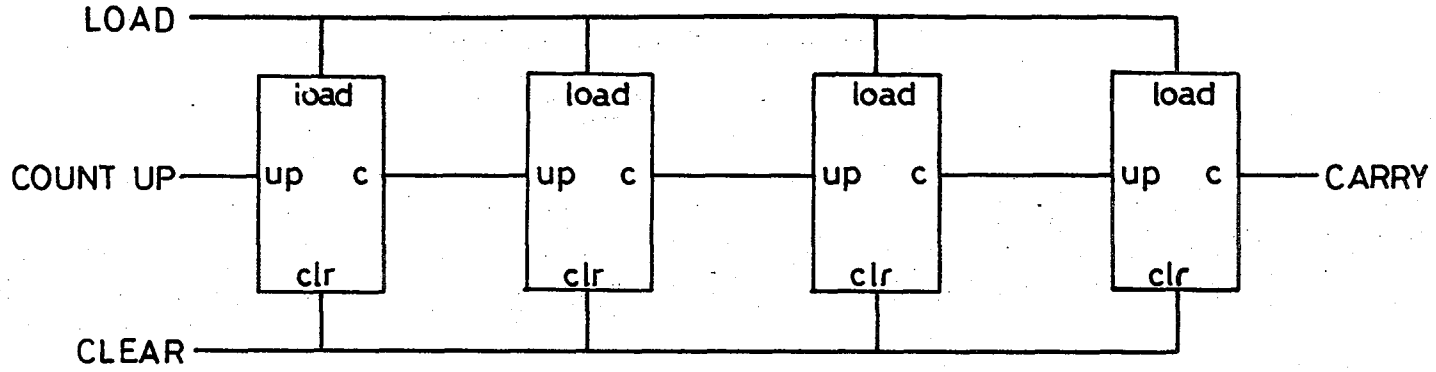
7.5.5 MBM Software Operation

The 7220 controller contains a 40 word FIFO that is used as a buffer when writing data to the MBM. Any write operation that allows the FIFO to become empty, or any read operation that allows the FIFO to become full will be aborted with a timing error. Thus the CPU must be able to keep up with the bubble memory to correctly read and write data. The status of the FIFO may be read from the 7220 status register. During a read operation a FIFO READY flag of '1' indicates that data is available to be read by the CPU, and during a write operation it indicates that more data is required.

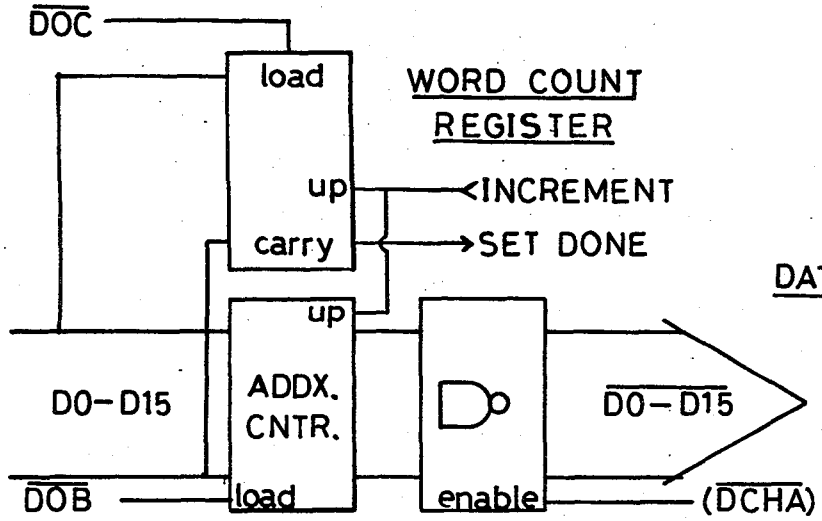
The data is written in a single page mode, that is the FIFO is supplied with, or will supply, 64 bytes of data before the controller sets the OP.COMPLETE flag in the status register indicating that the operation has been successful.

7.6 Conclusion

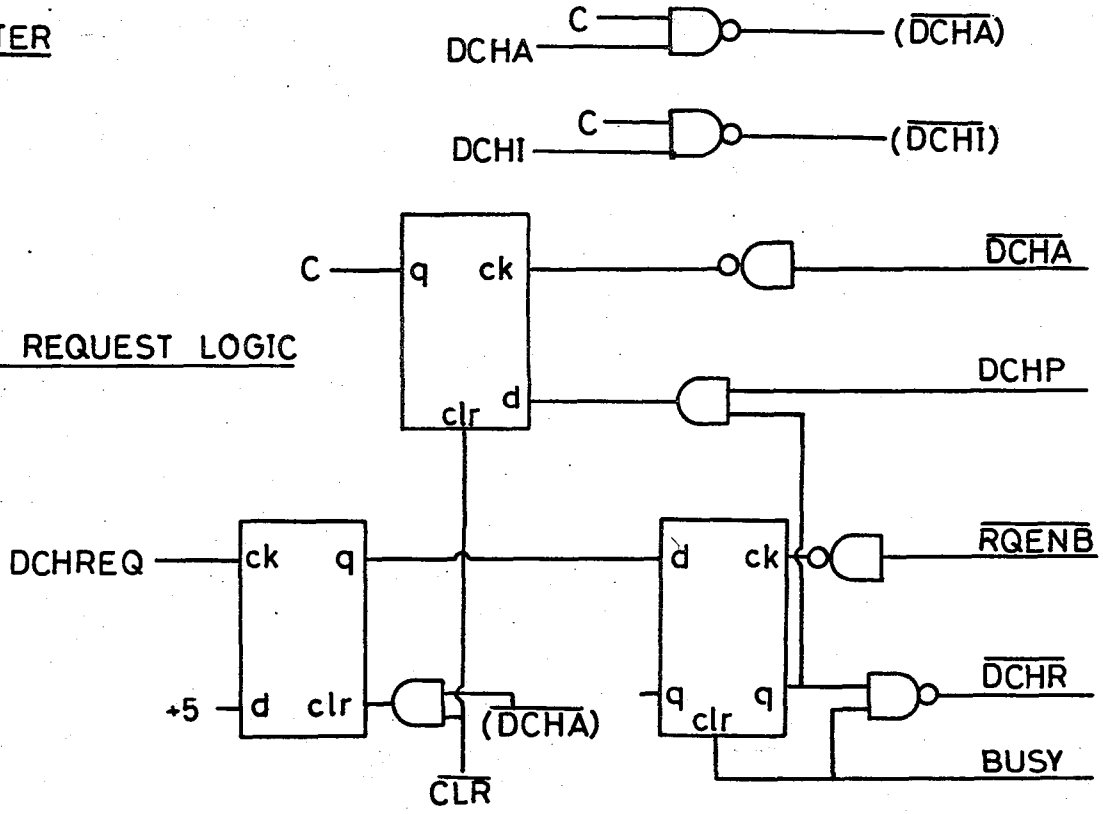
This chapter completes the description of the hardware and software used to construct the SABRE radar. The intention has been to provide the relevent information needed to understand the design. Specific details concerning manufacturers modules are contained in the publications listed in the references. With a computer system of this nature it is possible to discuss endlessly the advantages and disadvantages of design or equipment choice. The criterion used in the design of the hardware has been simplicity and maintainability. Undoubtably much of the small scale integration circuits used could now be incorporated into Programmable Logic Arrays (PLA), and these have the advantage of reducing the number of actual circuit packages considerably. Eventually one may envisage complete circuits incorporated into one or two Large Scale Integration (LSI) chips, especially as the techniques for producing them become easier and cheaper as the general trend shows. Until that time however, the boards full of logic for custom designed circuits, as is the case here, will persist.



16 BIT COUNTER

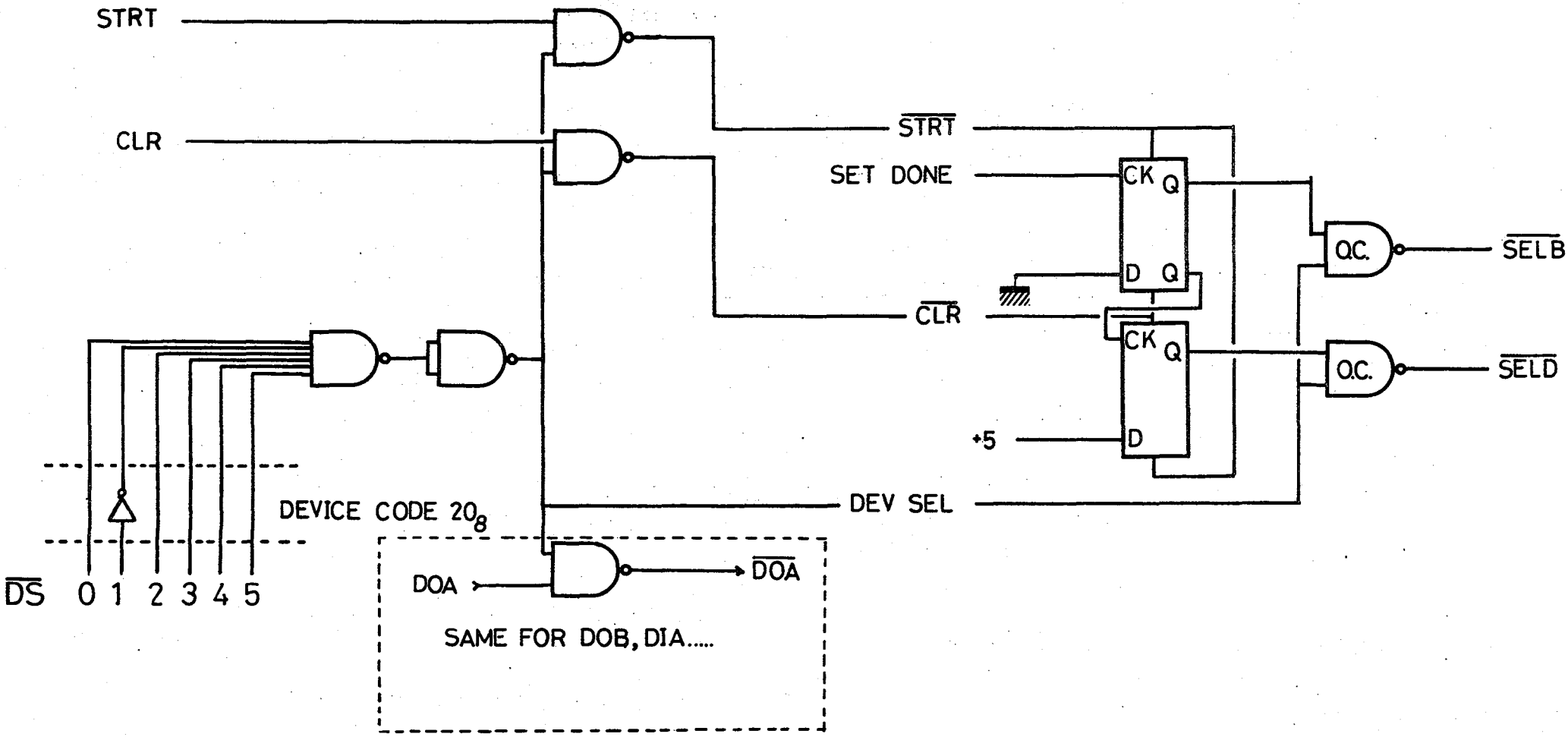


OPEN COLLECTOR BUS DRIVERS



DATA CHANNEL REQUEST LOGIC

FIGURE 7.1



DEVICE SELECT AND BUSY/DONE LOGIC

FIGURE 7.2

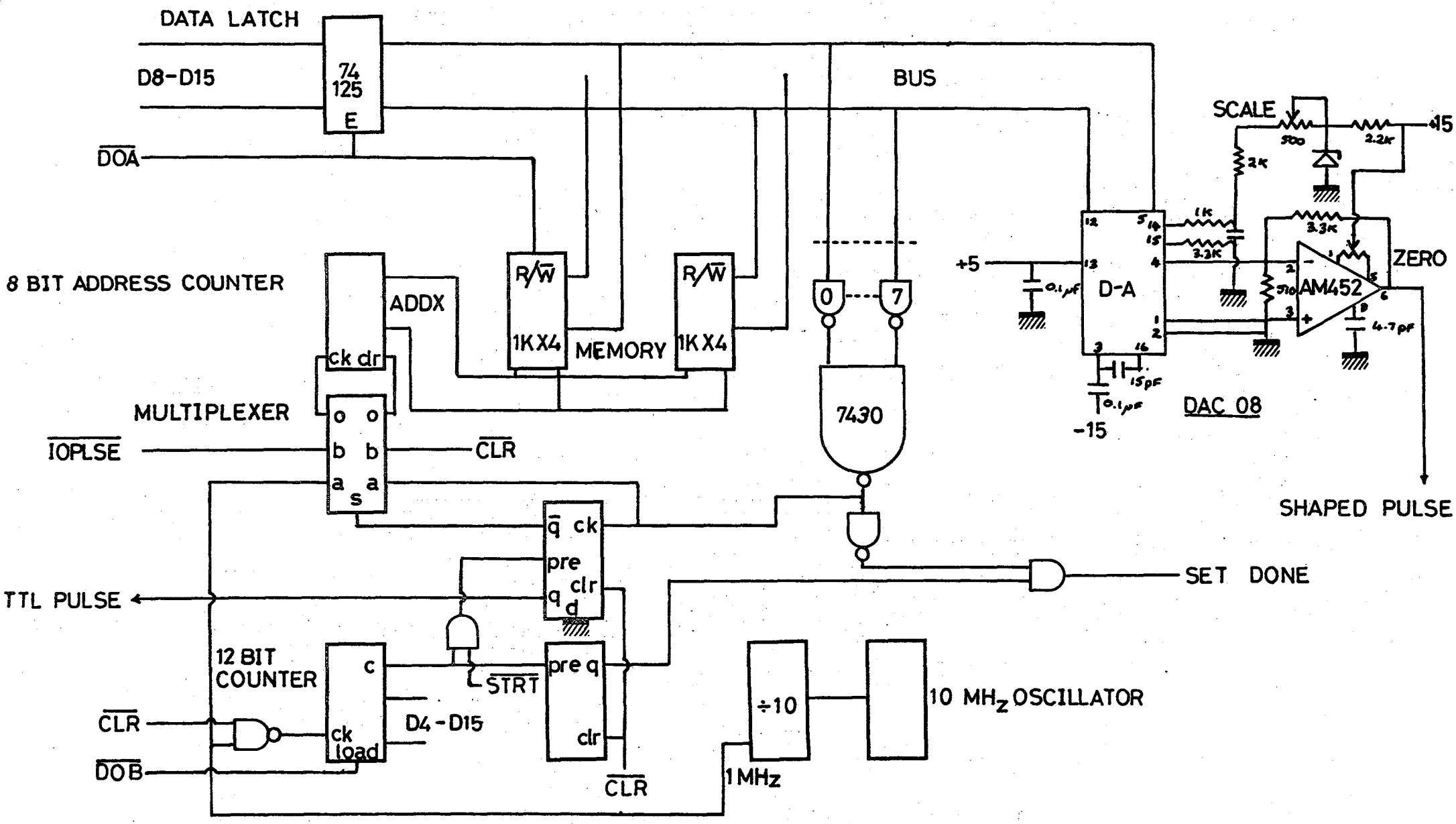

DATA LATCH
D8-D15
74
125
E
BUS
DOA
SCALE
500
2.2K
+15
2K
1K
3.3K
3.3K
ZERO
AM452
4.7pF
+5
0.1μF
D-A
15pF
0.1μF
-15
DAC 08
8 BIT ADDRESS COUNTER
ADDX
ck dr
R/W
1K X4
MEMORY
R/W
1K X4
0
7
7430
MULTIPLEXER
IOPLSE
CLR
SHAPED PULSE
SET DONE
TTL PULSE
12 BIT
COUNTER
c
ck
load
D4-D15
STRT
pre q
clr
CLR
÷10
1MHz
10 MHz OSCILLATOR
CLR
DOB

FIGURE 7.3

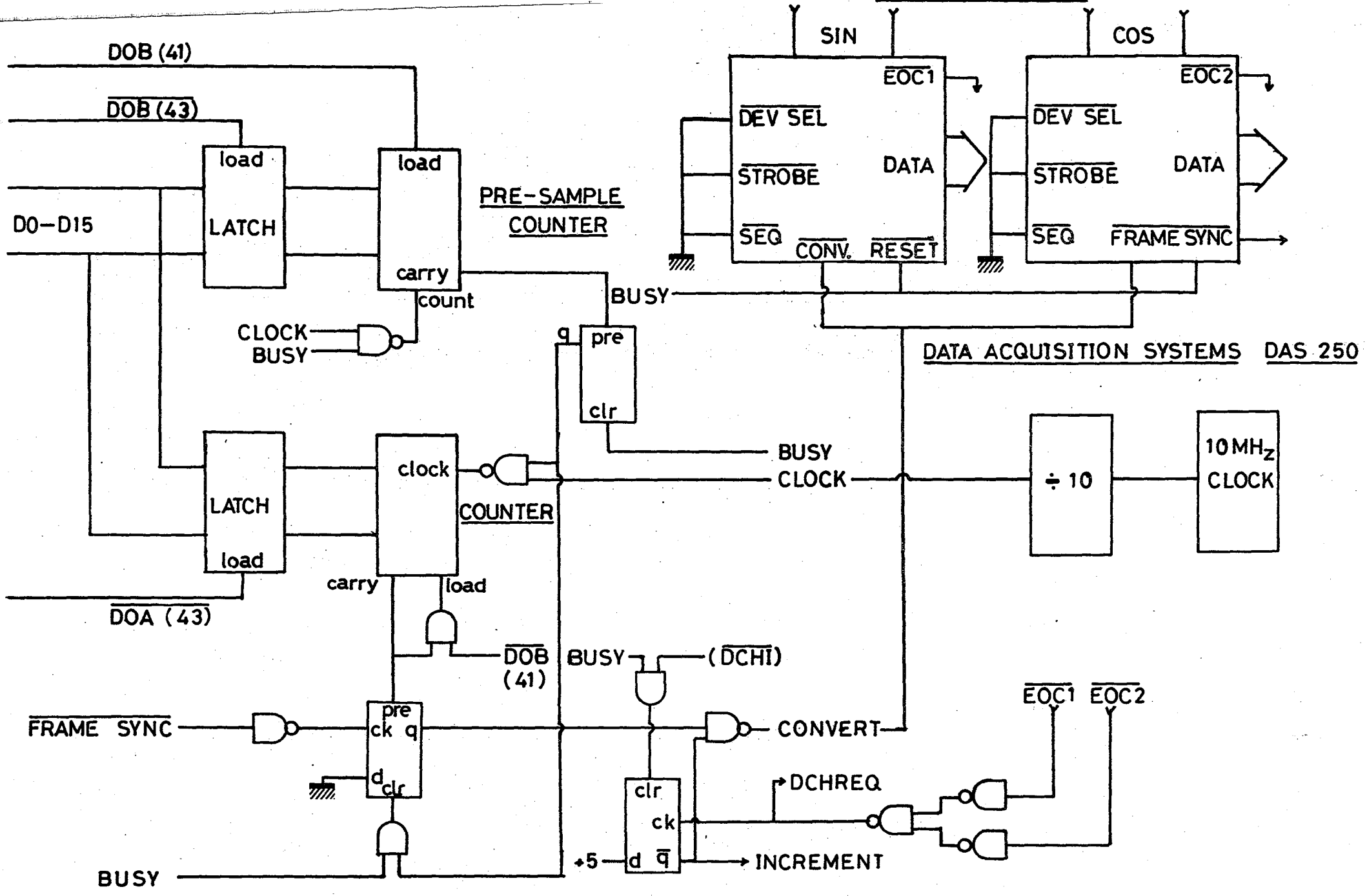
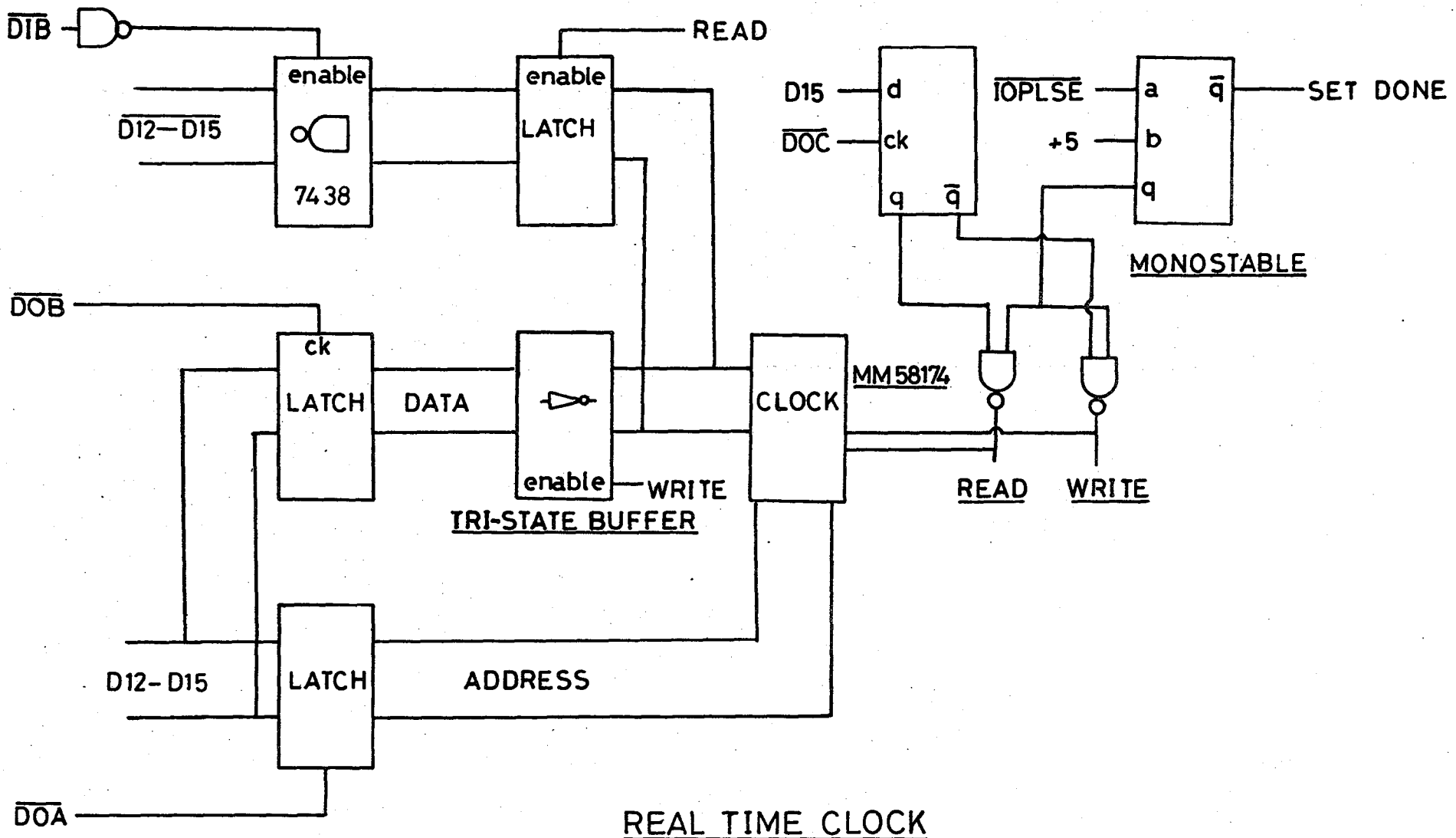

DOB (41)
DOB (43)
load
LATCH
load
carry
count
PRE-SAMPLE
COUNTER
D0–D15
CLOCK
BUSY
SIN
COS
EOC1
EOC2
DEV SEL
STROBE
DATA
SEQ
CONV.
RESET
FRAME SYNC
BUSY
DATA ACQUISITION SYSTEMS DAS 250
q
pre
clr
BUSY
CLOCK
÷ 10
10 MHz
CLOCK
LATCH
load
clock
COUNTER
carry
load
DOA (43)
DOB
(41)
BUSY
(DCHI)
EOC1 EOC2
FRAME SYNC
pre
ck q
d clr
CONVERT
DCHREQ
clr
ck
+5
d q̄
INCREMENT
BUSY

FIGURE 7.4



REAL TIME CLOCK

FIGURE 7.5

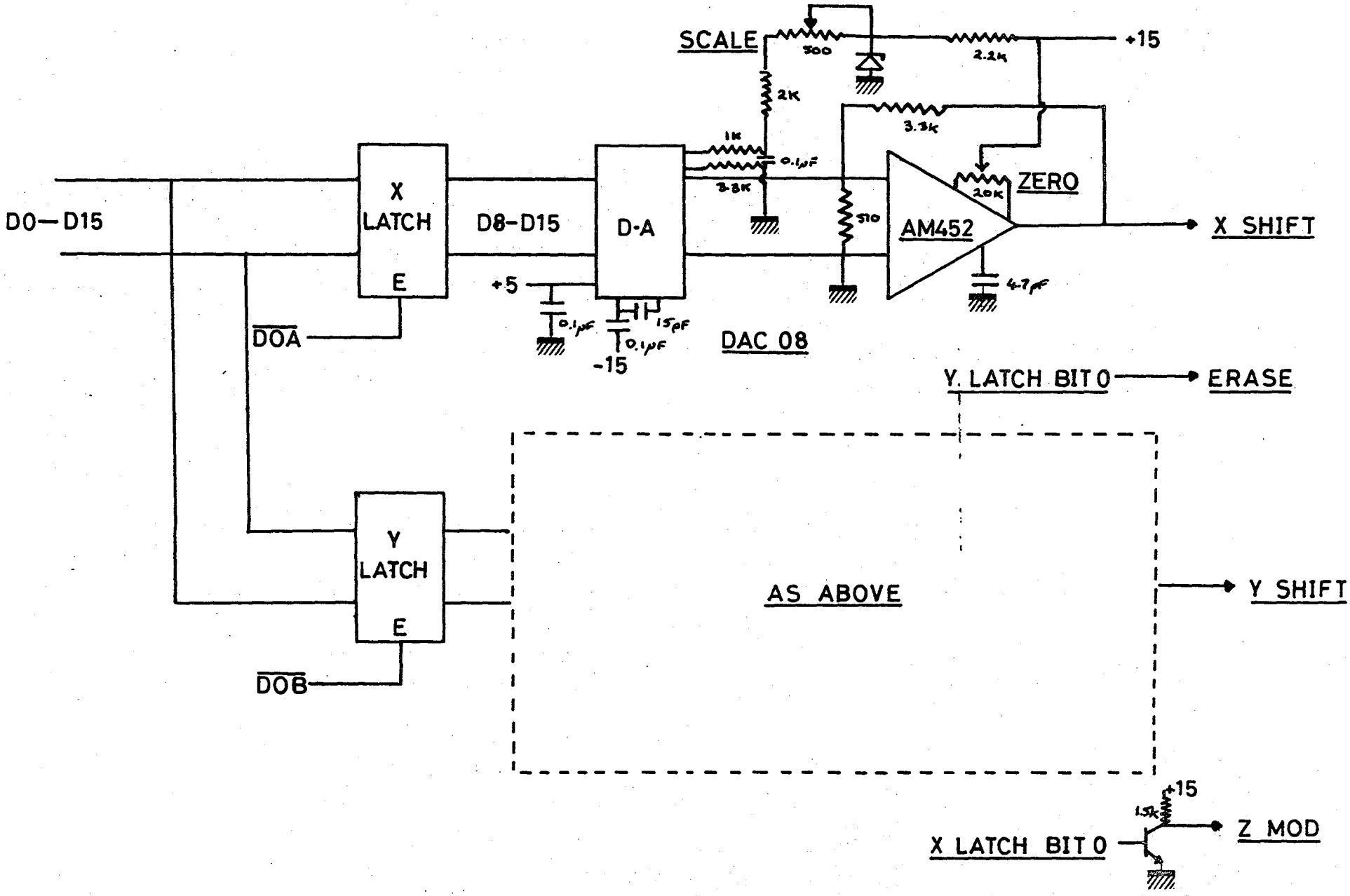
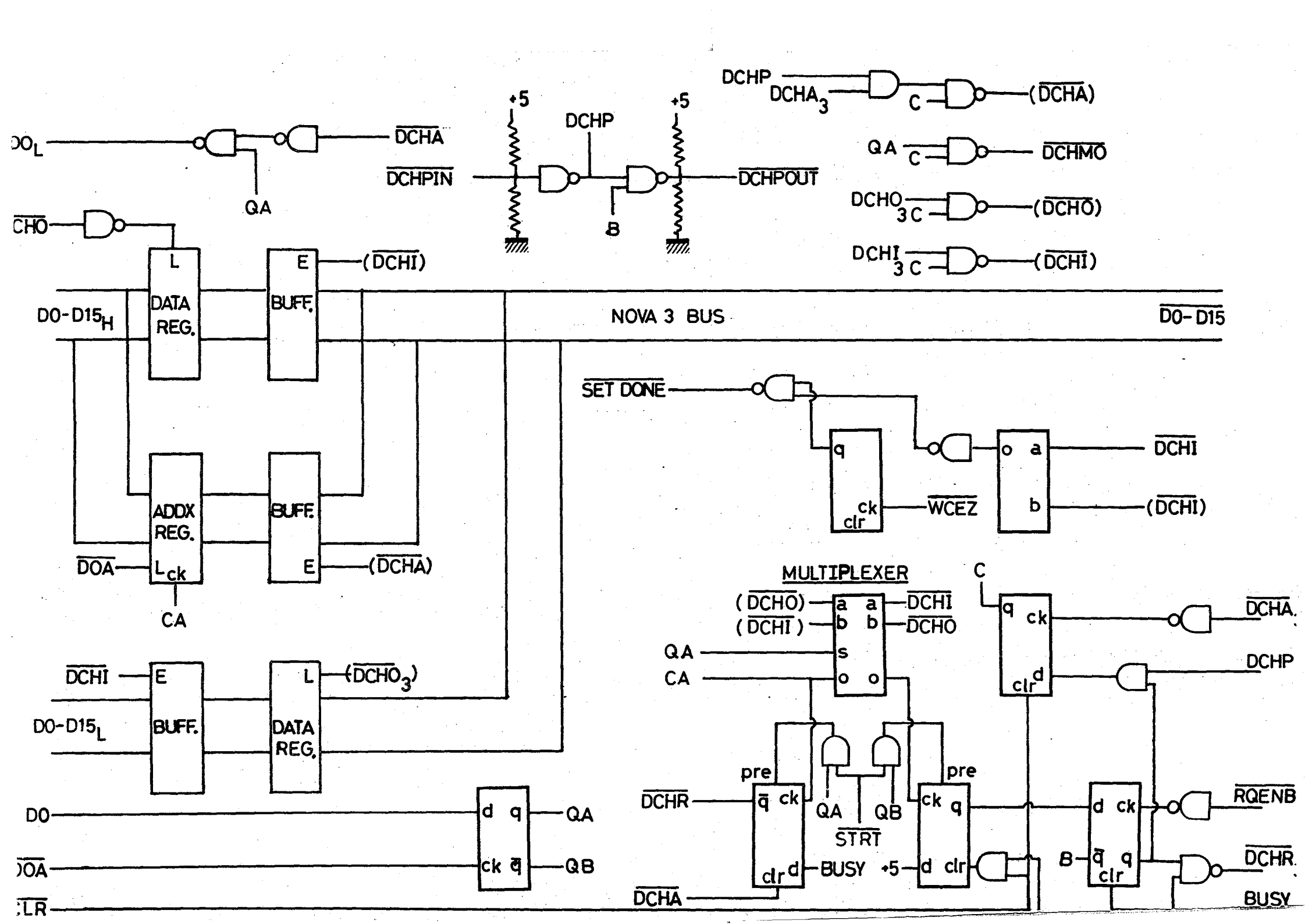
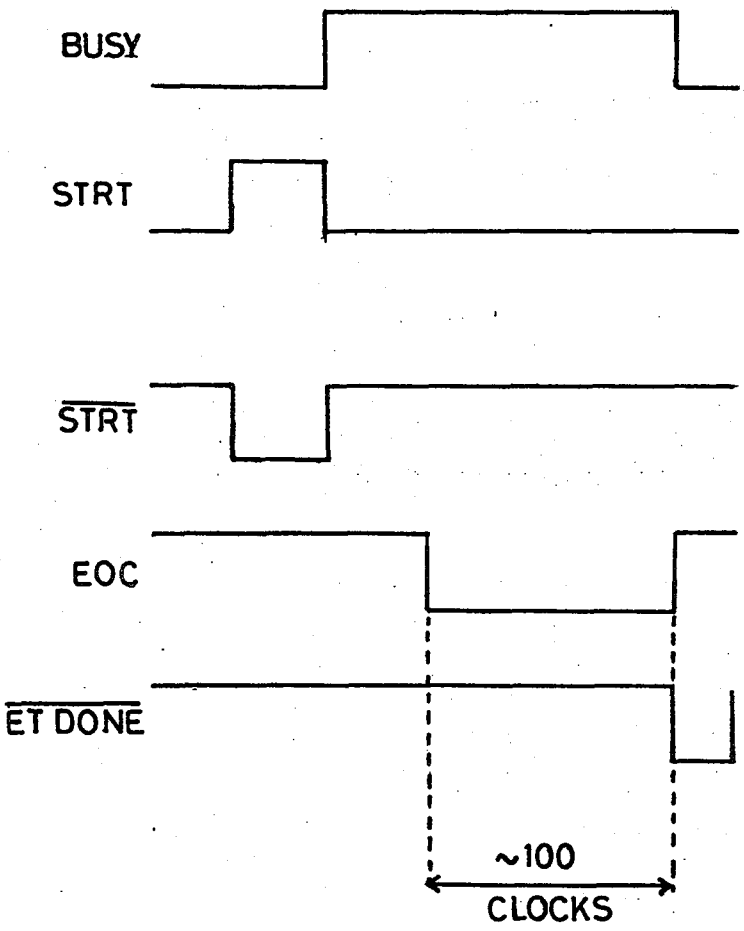
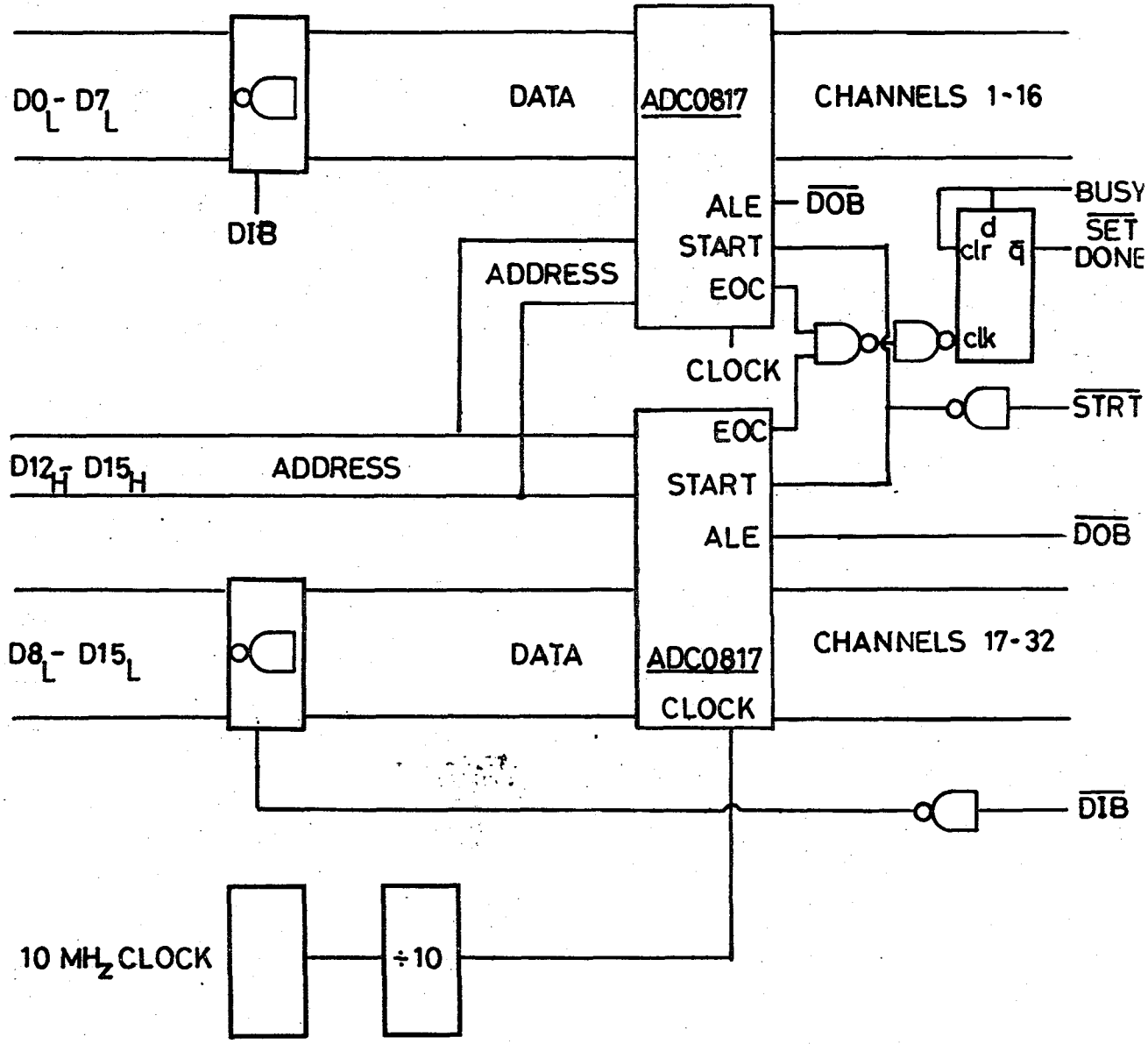

SCALE
300
2.2K
+15
2K
3.3K
1K
0.1μF
3.3K
ZERO
20K
X
LATCH
E
D0—D15
D8-D15
D-A
510
AM452
X SHIFT
+5
0.1μF
15pF
0.1μF
4.7μF
DOA
-15
DAC 08
Y. LATCH BIT 0
ERASE
Y
LATCH
E
AS ABOVE
Y SHIFT
DOB
+15
1.5K
Z MOD
X LATCH BIT 0

FIGURE 7.6


DCHP
DCHA3
C
(DCHA)
QA
C
DCHMO
DCHO
3C
(DCHO)
DCHI
3C
(DCHI)
+5
DCHP
+5
DCHPIN
DCHPOUT
B
QA
DCHA
L
DATA REG.
E
BUFF.
(DCHI)
D0-D15H
NOVA 3 BUS
D0-D15
ADDX REG.
BUFF.
DOA
Lck
E
(DCHA)
CA
SET DONE
q
ck
clr
WCEZ
o a
b
DCHI
(DCHI)
MULTIPLEXER
(DCHO)
(DCHI)
a a
b b
s
o o
DCHI
DCHO
QA
CA
C
q ck
clr d
DCHA
DCHP
DCHI
E
BUFF.
L
DATA REG.
(DCHO3)
D0-D15L
pre
DCHR
q ck
clr d
QA
QB
STRT
BUSY
+5
pre
ck q
d clr
d ck
B
q q
clr
RQENB
DCHR
BUSY
D0
d q
QA
DOA
ck q
QB
CLR
DCHA



TIMING DIAGRAM



HARDWARE MONITOR INTERFACE FIGURE 7.8

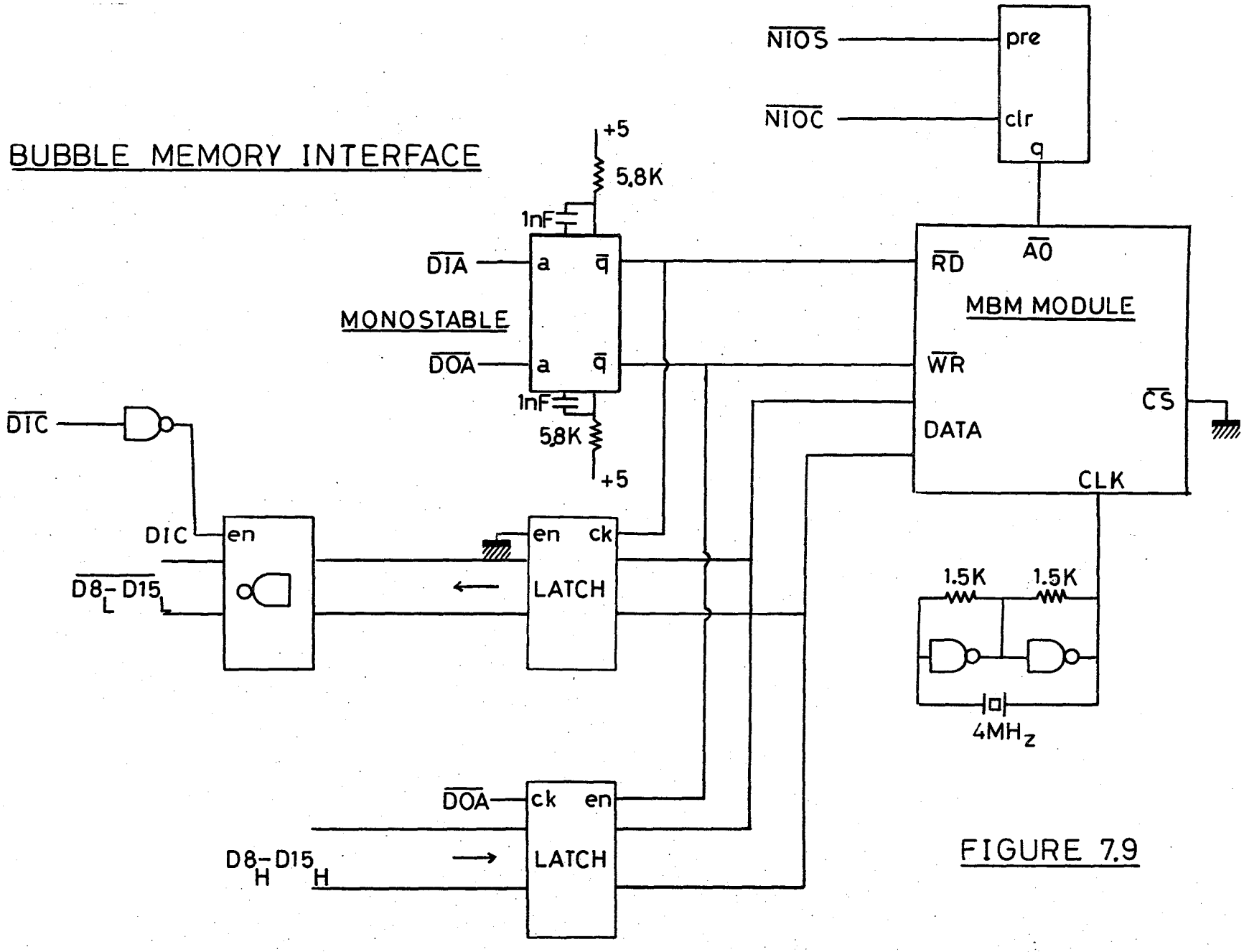

BUBBLE MEMORY INTERFACE
NIOS
NIOC
pre
clr
q
+5
5.8K
1nF
DIA
a
q
MONOSTABLE
DOA
a
q
1nF
5.8K
+5
RD
A0
MBM MODULE
WR
CS
DATA
CLK
DIC
en
en
ck
LATCH
D8L-D15L
1.5K
1.5K
4MHz
DOA
ck
en
LATCH
D8H-D15H

FIGURE 7.9

Chapter 8

SABRE Preliminary Results

8.0 Introduction

This chapter presents some of the preliminary results from the SABRE radar. The first few figures illustrate the fluctuations of some of the parameters, discussed in chapter 5, that are monitored by the Micronova computer. In each case the horizontal axis represents time, with the day number given underneath the midday hour number. The vertical scale in each case is a linear scale that is uncalibrated, except in the graphs showing the final power output (P.A.), where 200 represents a power output of 50 KW. Finally some range, time, intensity (RTI) plots are presented that show the intensity of the backscattered power over the viewing area as a function of time.

8.1 Monitor Data

Figure 8.1 shows the fluctuations of the power output during days 253 to days 258 inclusive. It can be seen that there are times during this period where no data is available. This is due to some short power cuts that occurred during this period. However the extent of the fluctuations of the output power during normal operation was quite surprising. At times the power output was reduced by as much as 50 percent. Figure 8.2 shows that the driver power output remained constant during this period, so that the output fluctuations are a sole function of the final stage of the transmitter. This fluctuation in radiated power may be attributed to the mis-match between the transmitter and antenna caused by high humidity weather conditions such as heavy rain or fog. Appreciation of this fact is important when

interpreting the radar backscattered power. Poor scattering may not be due to poor irregularity structure, but may be due to reduced power output caused by groung level atmospheric conditions. There is no reason why the integration time may not be altered automatically to compensate for reduced power output. This may be achieved simply by the Micronova altering the Nova 3 memory location that holds the integration time. The next time the Nova 3 calculates an end integration period, the new integration time would be used. The Finnish STARE radar is very susceptible to the effect of weather, and is often unable to provide data. The SABRE system is able to compensate for power fluctuations and provide data when otherwise it would be unavailable.

Figures 8.3 to 8.6 inclusive show an interesting period when the transmitter actually failed. This can be seen to have happened at 7.29 am on day 272. The failure was sudden, implying a component failure, that in fact turned out to be the EHT power supply.

8.2 Backscatter Data

The following plots show the backscatteded power as a function of time (horizontal scale) and range (vertical scale). A dark region indicates that the backscatter was above a threshold level, in this case 1.0 dB's. No attempt will be made to interpret, in terms of ionospheric physics, the data presented here, except to observe that significant ammounts of backscatter have been observed for extended periods (day 269-270) that indicate the quantity of data the SABRE radar will be able to provide.

8.3 Conclusion

This thesis has presented the details of the implementation of a scientific backscatter radar. It is hoped that some of the new ideas included in the SABRE system will be used in future auroral radar systems currently being considered (Greenwald, 1981). In particular, it has been proposed that many auroral radar may be connected via satellite to a large, central data analysis facility and perhaps provide both equatorial and auroral coverage of ionospheric conditions. Such a system is very ambitious but construction of systems such as SABRE provide an insight into the computational and design requirements that would be required for a network of real-time radar.

Appendix 1

Operating System Kernal

```
                        ;**************************************
                        ;* BYTE MANIPULATION ROUTINES         *
                        ;**************************************
                XBITE:  .TXT "(0)(4)BITE"
00000'000004
00001'041111
00002'052105
00003'000000
00004'000000 BBITE:     0
00005'075401 BITE:      PSHA 3          ; SAVE ALL ACC'S
00006'071401            PSHA 2
00007'065401            PSHA 1          ; ACC0=BYTE TO STORE
00010'026457            LDA 1,@.BS      ; LOAD BACKSPACE CHAR
00011'122434            SUBZ# 1,0,SZR   ; CURRENT CHAR=BACKSPACE ?
00012'000406            JMP NOTBS       ; NO
00013'024002-           LDA 1,START     ; YES SO MODIFY POINTER
00014'176520            SUBZL 3,3       ; GENERATE -1
00015'166400            SUB 3,1         ; BYTE POINTER-1
00016'044002-           STA 1,START     ; RESTORE IT
00017'000421            JMP ABORT       ; LEAVE ROUTINE
00020'024002-NOTBS:     LDA 1,START     ; GET CURRENT BYTE POINTER
00021'135220            MOVZR 1,3       ; 3=WORD POINTER
00022'021400            LDA 2,0,3       ; 2=WORD@3
00023'034445            LDA 3,MASK1     ; 3=177400
00024'125233            MOVZR# 1,1,SNC  ; CURRENT BYTE=1
00025'176400            SUB 3,3         ; NO SO MASK=0
00026'125233            MOVZR# 1,1,SNC  ; WHICH BYTE TO STORE?
00027'101300            MOVS 0,0        ; BYTE 0
00030'173420            ANDZ 3,2        ; FORCE APPROPRIATE BYTE(S)=0
00031'113020            ADDZ 0,2        ; ADD IN NEW BYTE
00032'135220            MOVZR 1,3       ; GET WORD POINTER AGAIN
00033'051400            STA 2,0,3       ; STORE NEW WORD
00034'125233            MOVZR# 1,1,SNC  ; COSMETIC ?
00035'101300            MOVS 0,0        ; YES SWAP BYTES BACK AGAIN
00036'125400            INC 1,1         ; BUMP BYTE POINTER
00037'044002-           STA 1,START     ; RESTORE IT
00040'065601 ABORT:     POPA 1          ; RETURN ALL THE
00041'071601            POPA 2          ; ACC'S
00042'075601            POPA 3          ; RETURN ADDRESS
00043'001400            JMP 0,3         ; GOODBYE
→ 0003  SABRE

                        ;**************************************
                        ;* GET ABYTE POINTED TO               *
                        ;**************************************
                XBYTE:  .TXT "(0)(4)BYTE"
00044'000004
00045'041131
00046'052105
00047'000000
00050'000000' BBYTE:    XBITE
00051'075401 BYTE:      PSHA 3          ; SAVE ACC'S
00052'065401            PSHA 1          ; USED
00053'024003-           LDA 1,GSTART    ; GET BYTE POINTER
00054'135220            MOVZR 1,3       ; GENERATE WORD POINTER
00055'021400            LDA 0,0,3       ; GOT WORD NOW
00056'125233            MOVZR# 1,1,SNC  ; WANT ODD OR EVEN ?
00057'101300            MOVS 0,0        ; EVEN
00060'034411            LDA 3,MASK2     ; 377
00061'163420            ANDZ 3,0        ; MASK OFF UNUSED BYTE
00062'125400            INC 1,1         ; BUMP BYTE POINTER
00063'044003-           STA 1,GSTART    ; RESTORE IT
00064'065601            POPA 1          ; RESTORE
00065'075601            POPA 3          ; ADDRESS
00066'001400            JMP 0,3         ; AND FLY
                        ;**************************************
                        ;* CONSTANTS USED BY THE ABOVE        *
                        ;**************************************
00067'001057'.BS:       BSPACE          ; BACKSPACE CHAR
00070'177400 MASK1:     177400          ; FOR TOP 8 BITS
00071'000377 MASK2:     377             ; FOR BOTTOM 8 BITS
→ 0004  SABRE
```

```
0001   SABRE
                        ;**************************************
                        ;*                                    *
                        ;* S.A.B.R.E OPERATING SYSTEM 1980    *
                        ;*  DAVID G. FOSTER 4 SEPT            *
                        ;*   UNIVERSITY OF DURHAM             *
                        ;*    DEPT APPLIED PHYSICS AND        *
                        ;*    ELECTRONICS, DURHAM, ENGLAND    *
                        ;*  REVIEW UPDATE 6 VERSION 8         *
                        ;*                                    *
                        ;**************************************
                        .TITLE SABRE
        000001          .TXTM 1
                        .NREL           ;NORMAL RELOCATABLE CODE
                        .ENT BEGIN
        061401          .DIAC PSHA=061401
        061601          .DIAC POPA=061601
        061001          .DIAC MTSP=061001
        061201          .DIAC MFSP=061201
        000034          .DUSR ULM=34
+ 0002  SABRE
```

```
                        ;**************************************
                        ;* NUMBER TRYS TO CONVERT CURRENT     *
                        ;* WORD TO A NUMBER AND FLAGS RESUL*
                        ;**************************************
                XNUMB:  .TXT "(0)(6)NUMB"
 00136'000006
 00137'047125
 00140'046502
 00141'000000
 00142'000116'BNUMB:    XMPY
 00143'075401 NUMB:     PSHA 3
 00144'071401           PSHA 2
 00145'024000-          LDA 1,WB          ; SET POINTERS TO WORD
 00146'044003-          STA 1,GSTART      ; WORD DONE
 00147'006474           JSR @.BY1         ; GET FIRST CHAR
 00150'024500           LDA 1,MINUS       ; GET ASCII MINUS
 00151'106405           SUB 0,1,SNR       ; SKIP IF NOT
 00152'006471           JSR @.BY1         ; GET ANOTHER CHAR
 00153'065401           PSHA 1            ; PUSH A FLAG=0 IF -VE <>0 IF +VE
 00154'126400           SUB 1,1           ; GENERATE A 0
 00155'065401           PSHA 1            ; RESERVE STACK SPACE FOR LATER
 00156'065401           PSHA 1            ; AND A BIT MORE
 00157'024472 INTO:     LDA 1,OFF1        ; FIRST ASCII OFFSET
 00160'122532           SUBZL# 1,0,SZC    ; CHECK < CHARACTER RANGE
 00161'000450           JMP TOQUE         ; INVOKE ERROR
 00162'122400           SUB 1,0           ; NOW SUBTRACT PROPERLY
 00163'024470           LDA 1,NINE        ; CHECK FOR NUMBER > 9
 00164'106533           SUBZL# 0,1,SNC    ; SKIP IF NUMBER > 9
 00165'000406           JMP TIME1         ; NOW JMP TO RADIX CHECK
 00166'024464           LDA 1,OFF2        ; SECOND OFFSET
 00167'106533           SUBZL# 0,1,SNC    ; CHECK FOR >9 <A
 00170'000441           JMP TOQUE         ; ERROR
 00171'024463           LDA 1,REL1        ; LOAD DIFFERENCE
 00172'122400           SUB 1,0           ; MODIFY SO WE HAVE A NUMBER
 00173'026454 TIME1:    LDA 1,@.RD1       ; NOW SEE IF THE BASE IS OK
 00174'106532           SUBZL# 0,1,SZC    ; WELL ?
 00175'000434           JMP TOQUE         ; ERROR
 00176'032451 TIMES:    LDA 2,@.RD1       ; SET UP TO MULTIPLY
 00177'065601           POPA 1            ; LAST NUMBER
 00200'006444           JSR @.MP1         ; MULTIPLY (ACC2*ACC1)+ACC0->ACC0,ACC1
 00201'075601           POPA 3            ; RECOVER LAST TOP BITS
 00202'065401           PSHA 1            ; SAVE NEW LEAST BITS
 00203'165000           MOV 3,1           ; GET READY FOR TOP BITS
 00204'006440           JSR @.MP1
 00205'075601           POPA 3            ; GET NEW LEAST BITS
 00206'065401           PSHA 1            ; SAVE NEW MOST BITS
 00207'075401           PSHA 3            ; SAVE NEW LEAST BITS
 00210'020001-          LDA 0,WE          ; CHECK FOR ENDWORD
 00211'024003-          LDA 1,GSTART      ; CURRENT WORD POINTER
 00212'122434           SUBZ# 1,0,SZR     ; LAST WORD ?
 00213'000426           JMP NEXT          ; NO SO REPEAT
 00214'061601           POPA 0            ; YES SO GET READY
 00215'065601           POPA 1            ; MOST SIG
 00216'071601           POPA 2            ; RECOVER STACK POINTER
 00217'151014           MOV# 2,2,SZR      ; WAS NUMBER -VE
 00220'000403           JMP PLUS          ; NO SO EXIT
 00221'100420           NEGZ 0,0          ; NEGATE NUMBERS
 00222'124000           COM 1,1
0007  SABRE
 00223'071601 PLUS:     POPA 2            ; GET STACK POINTER
 00224'006422           JSR @.PU1         ; PUSH LEAST 8 BITS
 00225'102520           SUBZL 0,0         ; GENERATE +1
 00226'042417           STA 0,@.NFG       ; STORE IN NUMBER FLAG
 00227'075601           POPA 3            ; RETURN
 00230'001400           JMP 0,3           ; BYE
 00231'061601 TOQUE:    POPA 0            ; CLEAN UP STACK
 00232'061601           POPA 0
 00233'061601           POPA 0            ; OK
 00234'071601           POPA 2            ; STACK POINTER
 00235'102400           SUB 0,0           ; CLEAR 0
 00236'042407           STA 0,@.NFG       ; FLAG NOT A NUMBER
 00237'075601           POPA 3            ; RETURN
 00240'001400           JMP 0,3           ; AND GO
 00241'006432 NEXT:     JSR @.BY1
 00242'000715           JMP INTO          ; ET NEXT BYTE AND RETURN
```

```
                        ;*****************************************
                        ;* SOFTWARE MULTIPLY/DIVIDE ROUTINE*
                        ;*****************************************
                XDIV:   .TXT "<0><3>DIV "
 00072'000003
 00073'042111
 00074'053040
 00075'000000
 00076'000044' BDIV:    XBYTE
 00077'075401  ADIV:    PSHA 3          ; SAVE ACC
 00100'142432           SUBZ# 2,0,SZC   ; DATA GENERAL ROUTINES
 00101'000412           JMP DIV1
 00102'034433           LDA 3,M20
 00103'125120           MOVZL 1,1
 00104'101100 DVD0:     MOVL 0,0
 00105'142412           SUB# 2,0,SZC
 00106'142400           SUB 2,0
 00107'125100           MOVL 1,1
 00110'175404           INC 3,3,SZR
 00111'000773           JMP DVD0
 00112'176441           SUBO 3,3,SKP
 00113'176420 DIV1:     SUBZ 3,3
 00114'075601           POPA 3          ; IN FPP MANUAL
 00115'001400           JMP 0,3 ; EXIT
→ 0005  SABRE

                        ;*****************************************
                        ;       AND MULTIPLY ROUTINE            *
                        ;*****************************************
                XMPY:   .TXT "<0><3>MUL "
 00116'000003
 00117'046525
 00120'046040
 00121'000000
 00122'000072' BMPY:    XDIV
 00123'075401  MPY:     PSHA 3          ; ALSO A.D.G. ROUTINE
 00124'034411           LDA 3,M20
 00125'125203  MPY1:    MOVR 1,1,SNC
 00126'101201           MOVR 0,0,SKP
 00127'143220           ADDZR 2,0
 00130'175404           INC 3,3,SZR
 00131'000774           JMP MPY1
 00132'125260           MOVCR 1,1
 00133'075601           POPA 3
 00134'001400           JMP 0,3         ; IN FPP MANUAL
                        ;*****************************************
                        ;*          BITS USED                    *
                        ;*****************************************
 00135'177760 M20:      -20
→ 0006  SABRE
```

```
                        ;**************************************
                        ;* THE NEW STACK POP ROUTINE OK       *
                        ;**************************************
                XUNSTK: .TXT "(0)(5)UNST"
   00314'000005
   00315'052516
   00316'051524
   00317'000000
   00320'000300'BUNSTK: XSTK
   00321'071401 UNSTK:  PSHA 2          ; SAVE OLD STACK POINTER
   00322'032771         LDA 2,@.STK2    ; GET NEW ONE
   00323'021000         LDA 0,0,2       ; GET WORD
   00324'012767         ISZ @.STK2      ; INCREMENT BUT DO NOT SKIP
   00325'071601         POPA 2          ; RECOVER OLD FAITHFUL
   00326'001400         JMP 0,3
→ 0012  SABRE

                        ;**************************************
                        ;*  GETS THE LENGTH OF CURRENT WORD*
                        ;**************************************
                XLEN:   .TXT "(0)(6)LENG"
   00327'000006
   00330'046105
   00331'047107
   00332'000000
   00333'000314'BLEN:   XUNSTK
   00334'075401 LEN:    PSHA 3          ; SAVE RETURN
   00335'024001-        LDA 1,WE        ; WORD END BYTE POINTER
   00336'020000-        LDA 0,WB        ; WORD BEGIN BYTE POINTER
   00337'106420         SUBZ 0,1        ; NO. OF BYTES
   00340'075601         POPA 3          ; EXIT
   00341'001400         JMP 0,3         ; BYE
→ 0013  SABRE
```

```
                        ;*****************************************
                        ;* THINGS USED BY NUMBER OK              *
                        ;*****************************************
00243'000051'.BY1:      BYTE
00244'000123'.MP1:      MPY
00245'001767'.NFG:      NFLAG
00246'000262'.PU1:      PUSH
00247'001765'.RD1:      RDX
00250'000055 MINUS:     55
00251'000060 OFF1:      60
00252'000020 OFF2:      20
00253'000011 NINE:      11
00254'000007 REL1:      7
→ 0008  SABRE

                        ;*****************************************
                        ;* OPERATIONAL STACK PUSH ROUTINE        *
                        ;*****************************************
                XPUSH:  .TXT "⟨0⟩⟨4⟩PUSH"
00255'000004
00256'050125
00257'051510
00260'000000
00261'000136'BPUSH:     XNUMB
00262'075401 PUSH:      PSHA 3          ; SAVE RETURN
00263'176520            SUBZL 3,3       ; GENERATE +1
00264'172420            SUBZ 3,2        ; DECREMENT STACK POINTER
00265'041000            STA 0,0,2       ; STORE RESULT
00266'075601            POPA 3          ; RECOVER
00267'001400            JMP 0,3         ; EXIT
→ 0009  SABRE

                        ;*****************************************
                        ;* OPERATIONAL STACK POP ROUTINE         *
                        ;*****************************************
                XPOP:   .TXT "⟨0⟩⟨3⟩POP "
00270'000003
00271'050117
00272'050040
00273'000000
00274'000255'BPOP:      XPUSH
00275'021000 POP:'      LDA 0,0,2       ; RECOVER WORD
00276'151400            INC 2,2         ; POINTER+1
00277'001400            JMP 0,3         ; EXIT
→ 0010  SABRE

                        ;*****************************************
                        ;* SECOND STACK PUSH ROUTIEN             *
                        ;*****************************************
                XSTK:   .TXT "⟨0⟩⟨3⟩STK "
00300'000003
00301'051524
00302'045440
00303'000000
00304'000270'BSTK:      XPOP
00305'071401 STK:       PSHA 2          ; SAVE USUAL POINTER
00306'016405            DSZ @.STK2      ; DECREMENT BUT NEVER SKIP
00307'032404            LDA 2,@.STK2    ; GET POINTER
00310'041000            STA 0,0,2       ; STORE WORD
00311'071601            POPA 2          ; RECOVER
00312'001400            JMP 0,3         ; BYE
                        ;*****************************************
                        ;* THE NEW STACK REFERENCE IS HERE       *
                        ;*****************************************
00313'001766'.STK2:     STK2
→ 0011  SABRE
```

```
                        ;**************************************
                        ;* DICTIONARY FIND ROUTINE            *
                        ;**************************************
                XFIN:   .TXT "(Ø)(4)FIND"
  00342'000004
  00343'043111
  00344'047104
  00345'000000
  00346'000327'BFIN:    XLEN
  00347'075401 FIN:     PSHA 3          ; SAVE POINTERS
  00350'071401          PSHA 2          ; STACK POINTER
  00351'006466          JSR @.LE2       ; LENGTH IN ACC1
  00352'030006-         LDA 2,DL        ; LAST DEFINITION
  00353'151005 AGAIN:   MOV 2,2,SNR     ; IFF ZERO =>  END OF DIC
  00354'000452          JMP QUIT        ; SO EXIT
  00355'021000          LDA Ø,Ø,2       ; GET LENGTH/PREC ENTRY
  00356'034462          LDA 3,MASK3     ; MASK OFF PREC
  00357'163400          AND 3,Ø         ; SO LENGTH OF DEFN IS IN ACCØ
  00360'106414          SUB# Ø,1,SZR    ; MATCH ?
  00361'000452          JMP AG2         ; NO SO GET NEXT DEFN
  00362'020000-         LDA Ø,WB        ; THE BYTES
  00363'040003-         STA Ø,GSTART    ; OK
  00364'006452          JSR @.BY2       ; GET A BYTE
  00365'115300          MOVS Ø,3        ; GET BYTE SWAP
  00366'075401          PSHA 3          ; SAVE BYTE SWAPPED
  00367'006447          JSR @.BY2       ; GET NEXT BYTE
  00370'176520          SUBZL 3,3       ; GENERATE +1
  00371'136415          SUB# 1,3,SNR    ; SEE IF LENGTH)1
  00372'020447          LDA Ø,SPA       ; LENGTH=1 FORCE BYTE2=SPACE
  00373'075601          POPA 3          ; RECOVER FIRST SWAPPED BYTE
  00374'117000          ADD Ø,3         ; NOW WE HAVE A COMPLETE WORD
  00375'021001          LDA Ø,1,2       ; SO GET WORD FROM HEADER
  00376'116414          SUB# Ø,3,SZR    ; COMPARE THEM
  00377'000434          JMP AG2         ; NOT EQUAL SO GET NEXT DEFN
  00400'102400          SUB Ø,Ø         ; GENERATE
  00401'101520          INCZL Ø,Ø       ; +2
  00402'122533          SUBZL# 1,Ø,SNC  ; LENGTH)2
  00403'000416          JMP GOTIT       ; NO SO WEVE FOUND IT
  00404'006432          JSR @.BY2       ; YES SO WE NEED TO LOOK MORE
  00405'115300          MOVS Ø,3        ; SWAP BYTE
  00406'075401          PSHA 3          ; SAVE IT
  00407'006427          JSR @.BY2       ; AND ANOTHER
  00410'176400          SUB 3,3         ; GENERATE
  00411'175540          INCOL 3,3       ; +3
  00412'136415          SUB# 1,3,SNR    ; SEE IF LENGTH)3
  00413'020426          LDA Ø,SPA       ; FORCE BYTE4=SPACE
  00414'075601          POPA 3          ; RECOVER SWAPPED BYTE
  00415'117000          ADD Ø,3         ; MAKE WORD
  00416'021002          LDA Ø,2,2       ; GET DEFINITION LAST WORD
  00417'116414          SUB# Ø,3,SZR
  00420'000413          JMP AG2         ; NO SO GET NEXT
  00421'141000 GOTIT:   MOV 2,Ø         ; OK WEVE GOT IT @ADDRESS=ACCØ
  00422'071601          POPA 2          ; RECOVER STACK POINTER
  00423'006412          JSR @.PU2       ; PUSH ADDRESS
  00424'075601          POPA 3          ; GET ADDRESS
  00425'001400          JMP Ø,3         ; BYE
  00426'071601 QUIT:    POPA 2          ; RECOVER STACK POINTER
  00427'102400          SUB Ø,Ø         ; FLAG ZERO ADDRESS
 0014  SABRE
  00430'006405          JSR @.PU2       ; PUSH IT
  00431'075601          POPA 3          ; RECOVER
  00432'001400          JMP Ø,3         ; EXIT
  00433'031004 AG2:     LDA 2,4,2       ; LOAD2 WITH LINK ADDRESS
  00434'000717          JMP AGAIN       ; TRY AGAIN
                        ;**************************************
                        ;* THINGS USED BY FIND IE ADDRESSES*
                        ;**************************************
  00435'000262'.PU2:    PUSH
  00436'000051'.BY2:    BYTE
  00437'000334'.LE2:    LEN
  00440'000377 MASK3:   377
  00441'000040 SPA:     40
* 0015  SABRE
```

```
                        ;*****************************************
                        ;* TYPE ROUTINE FOR N CHARS @ ADDX *
                        ;*****************************************
                XTYP:   .TXT "(0)(4)TYPE"
00442'000004
00443'052131
00444'050105
00445'000000
00446'000342'BTYP:      XFIN
00447'075401 TYP:       PSHA 3          ; SAVE ADDRESS
00450'006416            JSR @.PO3       ; POP NO OF CHARS TO TYPE
00451'105005            MOV 0,1,SNR     ; ZERO?
00452'000411            JMP NOTYPE      ; YES SO EXIT
00453'006413            JSR @.PO3       ; BYTE POINTER
00454'040003-           STA 0,GSTART    ; STORE IT IN BYTE ROUTINE
00455'071401            PSHA 2          ; SAVE STACK POINTER
00456'152520            SUBZL 2,2       ; GENERATE +1
00457'006410 MORE:      JSR @.BY3       ; GET A BYTE
00460'006410            JSR @.CO3       ; OUTPUT IT
00461'146404            SUB 2,1,SZR     ; COUNT DOWN
00462'000775            JMP MORE        ; AND AGAIN
00463'071601 NOTYP:     POPA 2          ; RECOVER
00464'075601            POPA 3          ; ADDRESS
00465'001400            JMP 0,3         ; AND LEAVE OK
                        ;*****************************************
                        ;* ADDRESSES REQUIRES BY TYPE    *
                        ;*****************************************
00466'000275'.PO3:      POP
00467'000051'.BY3:      BYTE
00470'001421'.CO3:      COUT
→ 0016  SABRE

                        ;*****************************************
                        ;*     THE TITLE ROUTINE    !       *
                        ;*****************************************
00471'001164"HERE:      .+1*2           ; THIS IS WHERE WE ARE
                        .TXT "(15)(12)↑SYS 8.6← (15)(12)"
00472'006412
00473'055523
00474'054523
00475'020070
00476'027065
00477'056440
00500'006412
00501'000000
                XTIT:   .TXT "(0)(5)TITL"
00502'000005
00503'052111
00504'052114
00505'000000
00506'000442'BTIT:      XTYP
00507'075401 TIT:       PSHA 3          ; SAVE RETURN ADDRESS
00510'020761            LDA 0,HERE      ; BYTE ADDRESS
00511'006407            JSR @.PU4       ; PUSH IT
00512'020405            LDA 0,TITLE     ; LENGTH
00513'006405            JSR @.PU4       ; PUSH IT
00514'006405            JSR @.TY4       ; TYPE IT
00515'075601            POPA 3          ; RECOVER
00516'001400            JMP 0,3         ; EXIT OK
                        ;*****************************************
                        ;*     THE USUAL BITS USED       *
                        ;*****************************************
00517'000016 TITLE:     16              ; NO OF CHARS IN TITLE
00520'000262'.PU4:      PUSH
00521'000447'.TY4:      TYP
→ 0017  SABRE
```

```
                        ;*************************************
                        ;* BITS AND PIECES USED BY BUFFER    *
                        ;*************************************
00610'000674'.CR6:      CRLF
00611'001762'.PR6:      PREFIX
00612'001421'.CO6:      COUT
00613'001445'.CI6:      CIN
00614'000005'.BI6:      BITE
00615'001057'.BSP:      BSPACE          ; BACKSPACE CHAR
00616'000015 CR:        15              ; CRETURN CHAR
00617'000037 NPRT:      37              ; NON PRINTING OFFSET
→ 0019  SABRE

                        ;*************************************
                        ;* SETS POINTERS TO CURRENT WORD     *
                        ;*************************************
                XWOR:   .TXT "(0)(4)WORD"
00620'000004
00621'053517
00622'051104
00623'000000
00624'000547'BWOR:      XBUFF
00625'075401 WOR:       PSHA 3          ; SAVE RETURN
00626'071401            PSHA 2          ; AND STACK POINTER
00627'030436            LDA 2,CRET      ; ASCII CRETURN
00630'024436            LDA 1,SPCE      ; ASCII SPACE
00631'020001-           LDA 0,WE        ; END OF LAST WORD
00632'040000-           STA 0,WB        ; =START OF THIS ONE
00633'040003-           STA 0,GSTART    ; WORD POINTER
00634'006427 ALONG:     JSR @.BY7       ; GET A BYTE
00635'106414            SUB# 0,1,SZR    ; A SPACE PERHAPS ?
00636'000404            JMP PICK        ; NO SO WE CAN START
00637'020003-           LDA 0,GSTART    ; GET CURRENT
00640'040000-           STA 0,WB        ; MOVE ALONG WORD START
00641'000773            JMP ALONG       ; NEXT PLEASE
00642'112434 PICK:      SUBZ# 0,2,SZR   ; CRETURN?
00643'000404            JMP GETIT       ; NO SO WE CARRY ON
00644'176520            SUBZL 3,3       ; GENERATE +1
00645'056417            STA 3,@.LAST    ; FLAG LAST WORD
00646'000406            JMP EXIT        ; AND LEAVE
00647'006414 GETIT:     JSR @.BY7       ; GET ANOTHER BYTE
00650'106415            SUB# 0,1,SNR    ; SPACE ?
00651'000403            JMP EXIT        ; YES GOODBYE
00652'112414            SUB# 0,2,SZR    ; CRETURN ?
00653'000774            JMP GETIT       ; NO SO CARRY ON FURTHER
00654'020003-EXIT:      LDA 0,GSTART    ; OVERSHOT POINTER BY 1 SO ADJUST BY -1
00655'126520            SUBZL 1,1       ; GENERATE +1
00656'122400            SUB 1,0         ; DONE
00657'040001-           STA 0,WE        ; STORE IN CURRENT WORD POINTER
00660'071601            POPA 2
00661'075601            POPA 3          ; EXIT
00662'001400            JMP 0,3         ; BYE BYE
                        ;*************************************
                        ;*   VALUABLE BITS AND PIECES        *
                        ;*************************************
00663'000051'.BY7:      BYTE
00664'001763'.LAST:     LAST
00665'000015 CRET:      15              ; CRETURN=15
00666'000040 SPCE:      40              ; ASCII SPACE
→ 0020  SABRE
```

```
                        ;*****************************************
                        ;* QUESTION AN UNIDENTIFIED WORD         *
                        ;*****************************************
                XQUE:   .TXT "(0)(10)QUES"
  00522'000010
  00523'050525
  00524'042523
  00525'000000
  00526'000502'BQUE:    XTIT
  00527'075401 QUE:     PSHA 3              ; SAVE RETURN ADDRESS
  00530'020000-         LDA 0,WB            ; START ADDRESS
  00531'006412          JSR @.PU5           ; PUSH IT
  00532'006410          JSR @.LE5           ; GET LENGTH
  00533'121000          MOV 1,0 ; MOVE TO 0 SO WE CAN PUSH IT
  00534'006407          JSR @.PU5           ; DONE
  00535'006407          JSR @.TY5           ; TYPE IT
  00536'020410          LDA 0,QU            ; "?"
  00537'006406          JSR @.CO5           ; TYPE (WORD)?
  00540'075601          POPA 3              ; RETURN ADDRESS
  00541'001400          JMP 0,3             ; EXIT
                        ;*****************************************
                        ;*.  ADDRESSES USED BY QUESTION         *
                        ;*****************************************
  00542'000334'.LE5:    LEN
  00543'000262'.PU5:    PUSH
  00544'000447'.TY5:    TYP
  00545'001421'.CO5:    COUT
  00546'000077 QU:      77                  ;"?"
→ 0018  SABRE

                        ;*****************************************
                        ;* I/O BUFFER ROUTINE FOR TERMINAL       *
                        ;*****************************************
                XBUFF:  .TXT "(0)(6)BUFF"
  00547'000006
  00550'041125
  00551'043106
  00552'000000
  00553'000522'BBUF:    XQUE
  00554'075401 BUF:     PSHA 3              ; SAVE ADDRESS
  00555'022434 PREF:    LDA 0,@.PR6         ; PREFIX CHAR
  00556'006434          JSR @.CO6           ; OUTPUT IT
  00557'020011-         LDA 0,STORE         ; SET START OF BUFFER
  00560'040002-         STA 0,START         ; FOR BITE ROUTINE
  00561'040001-         STA 0,WE            ; FOR WORD ROUTINE
  00562'006431 NCHAR:   JSR @.CI6           ; GET A CHAR
  00563'026432          LDA 1,@.BSP         ; GET BACKSPACE CHAR
  00564'106415          SUB# 0,1,SNR        ; CHECK CURRENT=BACKSPACE
  00565'000407          JMP OK              ; YES IT IS BUT IS OK
  00566'024430          LDA 1,CR            ; GET CR=15 OCTAL
  00567'106415          SUB# 0,1,SNR        ; CHECK CURRENT=CRETURN
  00570'000404          JMP OK              ; YES BUT IS OK
  00571'024426          LDA 1,NPRT          ; CHECK FOR ANYTHING ELSE
  00572'122532          SUBZL# 1,0,SZC      ; AND TRAP THEM
  00573'000767          JMP NCHAR           ; OK
  00574'006416 OK:      JSR @.CO6           ; ECHO CHAR
  00575'006417          JSR @.BI6           ; STORE IT
  00576'024002-         LDA 1,START         ; CHECK BEGINNING OF LINE
  00577'034011-         LDA 3,STORE         ; WITH STORE
  00600'166532          SUBZL# 3,1,SZC      ; CURRENT)BEGINNING
  00601'000754          JMP PREF            ; NO SO START AGAIN
  00602'024414          LDA 1,CR            ; CHECK FOR CRETURN AGAIN
  00603'122414          SUB# 1,0,SZR        ; WELL ?
  00604'000756          JMP NCHAR           ; NO SO GET NEXT
  00605'006403          JSR @.CR6           ; SEND CRLF
  00606'075601          POPA 3
  00607'001400          JMP 0,3             ; EXIT
```

```
                        ;**************************************
                        ;* MORE ODDS AND ENDS THAT WE NEED *
                        ;**************************************
 00762'000275'.PD9:     POP
 00763'001631'.AS9:     ASMB
 00764'000143'.NU9:     NUMB
 00765'001326'.CO9:     CONST
 00766'000527'.QU9:     QUE
 00767'000674'.CR9:     CRLF
 00770'000000 XFUR:     0
0022  SABRE



 00771'000005 FIVE:     5
 00772'177400 MSK1:     177400
 00773'006400 JSUB:     6400
+0023  SABRE
```

Appendix 2

ROS Source Code

```
: NACC WORD NUMBER 10 LEFT POP MOV 0 0 S Z PUSH + WORD NUMBER 10 LEFT + POP
ASMB ;
: I/O WORD NUMBER 15 LEFT + WORD NUMBER + POP ASMB ;
: NI/O WORD NUMBER + POP ASMB ;
: DOA IMMEDIATE 61000 I/O ;
: DOB IMMEDIATE 62000 I/O ;
: DOC IMMEDIATE 63000 I/O ;
: DIA IMMEDIATE 60400 I/O ;
: DIB IMMEDIATE 61400 I/O ;
: DIC IMMEDIATE 62400 I/O ;
: NIO IMMEDIATE 60000 NI/O ;
: ISKP IMMEDIATE 63400 NI/O ;
: LDA IMMEDIATE 20000 SACC ;
: STA IMMEDIATE 40000 SACC ;
: JMP IMMEDIATE 0 NACC ;
: ISZ IMMEDIATE 10000 NACC ;
: DSZ IMMEDIATE 14000 NACC ;
: 1+ 1 + ;
: DUP POP PUSH PUSH ;
: SWAP POP PSHA 0 POP MOV 0 1 POPA 0 PUSH MOV 1 0 PUSH ;
: LAND POP MOV 0 1 POP AND 1 0 Z PUSH ;
: = POP MOV 0 1 POP SUB 3 3 SUB 1 0 # SNR INC 3 3 MOV 3 0 ;
: ) POP MOV 0 1 POP SUB 3 3 SUB 0 1 Z L # SZC INC 3 3 MOV 3 0 ;
: ( POP MOV 0 1 POP SUB 3 3 SUB 1 0 Z L # SZC INC 3 3 MOV 3 0 ;
: BASE RDX @W ;
: - POP MOV 0 1 COM 1 1 Z SUB 0 0 INC 0 0 ADD 1 0 Z PUSH + ;
: 1- 1 - ;
: 1+ POP INC 0 0 PUSH ;
: →1- DUP @W 1- SWAP ! ;
: →1 DUP @W 1+ SWAP ! ;
: TO POP COUT ;
: SWAP PSHA 1 POP PSHA 0 POP MOV 0 1 POPA 0 PUSH MOV 1 0 PUSH POPA 1 ;
: DROP POP ;
: DUP POP PUSH PUSH ;
: OVER POP MOV 0 1 DUP POP PSHA 0 MOV 1 0 PUSH POPA 0 PUSH ;
: 4R MOV 0 0 R Z MOV 0 0 R Z MOV 0 0 R Z MOV 0 0 R Z ;
: 2L MOV 0 0 L Z MOV 0 0 L Z ;
: 6R 4R MOV 0 0 R Z MOV 0 0 R Z ;
: 4L 2L 2L ;
: 12R 6R 6R ;
: HERE DP @W ;
: => IMMEDIATE WORD FIND 5 + INTEGER ;
: CALL DP →1 6000 400 + DP @W ! ;
: <= CALL TABLE @W TPOINT @W + ! TPOINT →1 ;
: LAST@ DL @W ;
: $ IMMEDIATE WORD FIND 5 + <= ;
: =0SKIP 101024 POP ASMB 400 POP ASMB ;
: )0SKIP 101025 POP ASMB 400 POP ASMB ;
: IF IMMEDIATE )0SKIP HERE ;
: THEN IMMEDIATE DUP HERE 1 + SWAP - OVER @W POP MOV 0 1 POP ADD 1 0 Z PUSH SWAP
: ELSE IMMEDIATE 400 POP ASMB $ THEN HERE ;
: =( OVER OVER ( IF DROP DROP ELSE = THEN ;
: )= OVER OVER ) IF DROP DROP ELSE = THEN ;
: →LOOP STK2 @W 3 + STK2 ! SUB 0 0 Z ;
: (DO POP STK POP STK POP STK ;
: DO IMMEDIATE 0 INTEGER HERE => (DO <= ;
: (LOOP UNSTK PSHA 0 MOV 0 1 UNSTK
        SUB 0 1 Z INC 0 0 STK
                  POPA 0  STK
        MOV 1 0 Z SNR →LOOP ;
: LOOP IMMEDIATE => (LOOP <= =0SKIP
  HERE 2 - OVER ! 2 + HERE - 377 LAND 400 + HERE  !
      ;
: ABORT →LOOP POPA 0  ;
: NEXT POPA 0 UNSTK PSHA 0 UNSTK PSHA 0 UNSTK PUSH
              STK POPA 0   STK POPA 0   STK POP PSHA 0 ;
: STOP POPA 0 UNSTK UNSTK UNSTK PUSH 3 + POP PSHA 0 ;
: I UNSTK PSHA 0 UNSTK PUSH STK POPA 0 STK ;
: BEGIN IMMEDIATE HERE 1 + ;
```

```
        .END BEGIN
 28 CONSTANT  WB
 WB 1 + CONSTANT  WE
 WE 1 + CONSTANT  ST
 ST 1 + CONSTANT  GT
 GT 1 + CONSTANT STATE
STATE 1 + CONSTANT ERROR
ERROR 1 + CONSTANT DL
DL 1 + CONSTANT TABLE
TABLE 1 + CONSTANT TPOINT
TPOINT 1 + CONSTANT STORE
STORE 1 + @W CONSTANT OPSTK
OPSTK 1 + CONSTANT LOSTK
LOSTK 1 + CONSTANT PREFIX
PREFIX 1 + CONSTANT LAST
LAST 1 + CONSTANT REFLAG
REFLAG 1 + CONSTANT RDX
RDX 1 + CONSTANT STK2
STK2 1 + CONSTANT NFLAG
NFLAG 1 + CONSTANT INTERPRET
14 CONSTANT DP
: OCT 8 RDX ! ; : HEX 10 RDX ! ; : BIN 2 RDX ! ;
: DEC A RDX ! ;
OCT
: ACC WORD NUMBER 13 LEFT + POP ASMB ;
: DACC WORD NUMBER 15 LEFT + WORD NUMBER 13 LEFT + POP ASMB ;
: MUL IMMEDIATE 73301 POP ASMB ;
: DIV IMMEDIATE 73101 POP ASMB ;
: POPA IMMEDIATE 61601 ACC ;
: PSHA IMMEDIATE 61401 ACC ;
: MFSP IMMEDIATE 61201 ACC ;
: MTSP IMMEDIATE 61001 ACC ;
: ADD IMMEDIATE 103000 DACC ;
: SUB IMMEDIATE 102400 DACC ;
: NEG IMMEDIATE 100400 DACC ;
: ADC IMMEDIATE 102000 DACC ;
: MOV IMMEDIATE 101000 DACC ;
: INC IMMEDIATE 101400 DACC ;
: COM IMMEDIATE 100000 DACC ;
: AND IMMEDIATE 103400 DACC ;
: FRAME IMMEDIATE 62401 POP ASMB ;
: RETURN IMMEDIATE 62601 POP ASMB ;
: MOD DP @W @W +.DP @W ! ;
: @ IMMEDIATE 2000 MOD ;
: SKP IMMEDIATE 1 MOD ;
: SZC IMMEDIATE 2 MOD ;
: SNC IMMEDIATE 3 MOD ;
: SZR IMMEDIATE 4 MOD ;
: SNR IMMEDIATE 5 MOD ;
: SEZ IMMEDIATE 6 MOD ;
: SBN IMMEDIATE 7 MOD ;
: # IMMEDIATE 10 MOD ;
: Z IMMEDIATE 20 MOD ;
: O IMMEDIATE 40 MOD ;
: C IMMEDIATE 60 MOD ;
: L IMMEDIATE 100 MOD ;
: R IMMEDIATE 200 MOD ;
: S IMMEDIATE 300 MOD ;
: BN IMMEDIATE 0 MOD ;
: BZ IMMEDIATE 100 MOD ;
: DN IMMEDIATE 200 MOD ;
: DZ IMMEDIATE 300 MOD ;
: SR IMMEDIATE 100 MOD ;
: CL IMMEDIATE 200 MOD ;
: PU IMMEDIATE 300 MOD ;
: SACC WORD NUMBER 13 LEFT + WORD NUMBER 10 LEFT POP MOV 0 0 S Z PUSH +
WORD NUMBER 10 LEFT + POP ASMB ;
```

```
: END IMMEDIATE HERE 1 + - 377 LAND 400 + POP ASMB ;
: LOOK WB @W GT ! ;
: PUT DP @W 1+ POP MOV 0 0 L Z PUSH ST ! ;
: () OVER OVER ( IF SWAP THEN ;
: ( IMMEDIATE LOOK 100 1 DO BYTE PUSH 51 = IF
      GT @W WE ! ABORT THEN LOOP ;
: ALLOCATE DP @W + DP ! ;
: DELIM WORD LOOK BYTE PUSH ;
: WIDTH VARIABLE ;
: DP! DP @W 1+ DUP DP ! ! ;
: LEAP POPA 3 LDA 0 0 3 PSHA 3 PUSH
      POPA 3 LDA 0 1 3 PSHA 3 PUSH
      POPA 3 MOV 3 0 PUSH 2 + OVER + POP MOV 0 3 JMP 0 3 ;
: FILL 0 WIDTH ! DP @W 4 + 1 LEFT ST ! DELIM
   100 1 DO BYTE PUSH OVER OVER = IF DROP DROP WIDTH @W GT @W WE ! ABORT THEN
         WIDTH +1 POP BITE LOOP ;
: ARRAY =) LEAP (= DP @W 3 + DP! DUP DP! ALLOCATE ;
: *ARRAY $ : ARRAY $ ; ;
: SAY SWAP POP MOV 0 0 Z L PUSH SWAP TYPE ;
: STRING IMMEDIATE FILL ARRAY ;
: DIGIT POP MOV 0 1 RDX @W POP PSHA 2 MOV 0 2 SUB 0 0 DIV POPA 2 PUSH MOV 1 0
PUSH ;
: NUMERIC 1 DO DUP 11 ) IF 67 + TO ELSE 60 + TO THEN LOOP ;
: PAUSE PSHA 0 CIN CIN POPA 0 ;
: VDU ISKP 10 DZ PAUSE ;
: DISSECT VDU 20 1 DO DIGIT DUP 0 = IF DROP I ABORT THEN LOOP ;
: _ DISSECT NUMERIC ;
: 0. DISSECT RDX @W 2 = IF 20
     ELSE    RDX @W 10 = IF 6
     ELSE    RDX @W 12 = IF 5
     ELSE    RDX @W 20 = IF 4
THEN THEN THEN THEN
OVER OVER )= IF DROP ELSE OVER - 1 DO 60 TO LOOP THEN NUMERIC ;
: . DUP 0 )= IF ELSE 55 TO POP NEG 0 0 Z PUSH THEN _ ;
: , CRLF . ;
: FROM CIN PUSH ;
: * POP MOV 0 1 POP PSHA 2 MOV 0 2 SUB 0 0 MUL POPA 2 MOV 1 0 PUSH ;
: / POP MOV 0 1 POP PSHA 2 MOV 0 2 SUB 0 0 DIV POPA 2 MOV 1 0 PUSH ;
: NAME 2 1 DO DUP I + @W DUP POP 4R 4R COUT POP MOV 0 0 Z S 4R 4R COUT
           LOOP DROP ;
: BIT PUSH SWAP LEFT POP SUB 1 1 MOV 0 0 Z L # SZC INC 1 1 MOV 1 0 ;
: SPACE 40 TO ;
: SPACES 1 DO SPACE LOOP ;
: LOC WORD FIND ;
: LIST DO I @W . SPACE LOOP ;
: DEFN DUP @W SPACE 177400 POP MOV 0 1 POP  AND 1 0 S PUSH .
       DUP @W SPACE 377 POP MOV 0 1 POP AND 1 0 PUSH . ;
: .K VARIABLE ;
: KEEP WORD FIND .K ! ;
: FORGET WORD FIND DUP 0 = IF
   CRLF DROP STRING '*NOT FOUND*' SAY ELSE DUP .K @W =( IF
   CRLF DROP STRING '*DEFINITION KEPT*' SAY ELSE DUP 1- DP ! 4 + @W DL !
   THEN THEN ;
: WHAT CRLF DL @W 1000 1 DO
DUP DUP 0. SPACE NAME DEFN DUP .K @W = IF DROP ABORT THEN
4 + @W CRLF DUP 0 = IF
DROP STRING 'THERE ARE ' SAY I DEC . OCT STRING ' DEFINITIONS' SAY
ABORT THEN LOOP ;
: SP MOV 2 0 PUSH ;
: REBOOT POPA 0 INTERPRET @W 1- POP PSHA 0 ;


( ********************************************** )
( * COMMUNICATIONS LINK NOVA3/MICRONOVA        * )
( ********************************************** )
: 3LOOP ISKP 50 DN JMP 5 1 DIA 0 50 CL ISKP 11 BZ
        JMP -1 1 DOA 0 11 SR POPA 0 PUSH 2 - POP PSHA 0 ;
```

```
: DMAIN ISKP 10 DN 3LOOP DIA 0 10 CL
      PUSH DUP 33 = IF DROP JMP 7 1 THEN POP
      ISKP 51 BZ JMP -1 1 .DOA 0 51 SR POPA 0 DMAIN ;
: TALK CRLF STRING '↑MICRO (=) NOVA.3 LINK←' SAY CRLF DMAIN ;
( ********************************************** )
( * SYSTEM INTEGRITY CHECK ROUTINES            * )
( ********************************************** )
: RUN -1 POP DOA 0 60 CL 0 POP DOB 0 60 0 POP DOC 0 60 SR ;
: MEM VARIABLE ;
: MAP POP DOA 0 60 CL MEM POP DOB 0 60 -1 POP DOC 0 60 SR ;
: 0MAP 1- DUP 0 < IF ELSE 100000 + THEN MAP ;
: MEM_MAP 77777 0 DO I MAP ISKP 60 BZ JMP -1 1
        I @W MEM @W = IF ELSE I 0. SPACE I @W 0. SPACE MEM @W 0. CRLF THEN LOOP
( ********************************************** )
( * POWER FAIL AUTO RESTART CAUSES NOVA 3 LO   * )
( * LOOP ON LOCATION ZERO UNTIL RELOAD COMPLETE )
( * THEN EXECUTES A JMP @ 300                 * )
( ********************************************** )
: POWER_FAIL 0 MEM ! 77777 0 DO I 0MAP ISKP 60 BZ JMP -1 1 LOOP
            2373 300 ! RUN ISKP 60 BZ JMP -1 1 2300 MEM ! 0 0MAP
            ISKP 60 BZ JMP -1 1 ;
( ********************************************** )
( *  TAPE DRIVING PROGEAMS EMI 8800 S DECK     * )
( ********************************************** )
: TADX IMMEDIATE 62022 POP ASMB ; ( TAPE ADDRESS )
: 2'S POP NEG 0 0 Z ;             ( 2'S COMP     )
: COUNT IMMEDIATE 63022 POP ASMB ;
: STATUS IMMEDIATE 60422 POP ASMB ;
( ############################################## )
( #  COMMANDS TO THE TAPE DECK                 # )
( ############################################## )
: COMMAND IMMEDIATE 61122 POP ASMB ;
: REWIND 10 POP COMMAND ;
: FSPACE 30 POP COMMAND ;
: BSPACE 40 POP COMMAND ;
: WRITE  50 POP COMMAND ;
: ERASE  70 POP COMMAND ;
: READ   00 POP COMMAND ;
( ############################################## )
( #  ERROR MANAGMENT ROUTINES                  # )
( ############################################## )
: MSG0 STRING 'DECK ERROR =' CRLF SAY STATUS PUSH 0. CRLF ;
: MSG1 STRING 'TAPE UNIT 22 NOT READY' CRLF SAY CRLF ;
: MSG2 STRING 'END OF TAPE ' CRLF SAY CRLF ;
: MSG3 STRING 'START OF TAPE ' CRLF SAY CRLF ;
: ERROR? 0 STATUS BIT ;
: EOT? 6 STATUS BIT ;.
: BOT? 10 STATUS BIT ;
: READY? 17 STATUS BIT ;
: TBUSY? ISKP 22 BZ JMP -1 1 ;
: TDONE? ISKP 22 DN JMP -1 1 ;
( ############################################## )
( # COMMAND MACROS FOR TAPE UNIT 22           # )
( ############################################## )
: RWAIT TBUSY? READY? IF ELSE MSG1 READY? IF ELSE JMP -4 1 THEN THEN ;
: FOREWARD EOT? IF MSG2 ELSE 2'S COUNT FSPACE ERROR? IF MSG0 THEN ;
: BACKWARD BOT? IF MSG3 ELSE 2'S COUNT BSPACE ERROR? IF MSG0 THEN ;
: FWD RWAIT FOREWARD ;
: BWD RWAIT BACKWARD ;
: TREAD POP TADX 2'S COUNT READ ERROR? IF MSG0 THEN ;
: TWRITE POP TADX 2'S COUNT WRITE ERROR? IF MSG0 THEN ;
: READ RWAIT TREAD ;
: WRITE RWAIT TWRITE ;
( ********************************************** )
( *  TAPE EXERCISER FOR TAPE UNIT              * )
( ********************************************** )
: *R 1000 34000 READ ;
: *W 1000 34000 WRITE ;
: NWRITE 34777 34000 DO DUP I ! LOOP DROP ;
: ERROR VARIABLE ;
: NREAD 34777 34000 DO DUP I @W = IF ELSE ERROR +1 THEN LOOP DROP ;
```

```
( ************************************ )
( * USER INTERFACE SOFTWARE          * )
( ************************************ )
101 CONSTANT INTTIM
INTTIM 1 + CONSTANT NAVE
106 CONSTANT NRANG
NRANG 1 + CONSTANT SDLY
SDLY 2 + CONSTANT NBEAM
134 CONSTANT PGAP
PGAP 1 + CONSTANT DVALS
177 CONSTANT DBASE
213 CONSTANT PULSE
PULSE 1 + CONSTANT PLENG
PLENG 1 + CONSTANT MICRO
MICRO 1 + CONSTANT NOVA
240 CONSTANT SORT
SORT 1 + CONSTANT ADX3
ADX3 1 + CONSTANT MADX
MADX 1 + CONSTANT NWDS
( ************************************ )
( * THE RADAR PARAMETERS ON ZERO PAGE* )
( ************************************ )
: INTRO STRING 'The current radar parameters are as follows : ' SAY CRLF ;
: 1NAVE STRING 'The no. of averages so far is ' SAY ;
: 1INTT STRING 'The integration time (x 10 seconds) is ' SAY ;
: 1NBEA STRING 'The number of beams used are ' SAY ;
: 1NRAN STRING 'The number of ranges used are ' SAY ;
: 1PGAP STRING 'The number of samples in double pulse gap is ' SAY ;
: MSAVE SWAP MEM ! 0MAP ;
: MLOOK MAP MEM @W SPACE . CRLF ;
: PARAMETERS CRLF INTRO
  1NAVE NAVE MLOOK
   1INTT INTTIM MLOOK
  1NBEA NBEAM MLOOK
  1NRAN NRANG MLOOK
  1PGAP PGAP MLOOK CRLF ;
: DISPLAY STRING 'The current display is of ' SAY SORT MAP MEM @W
0 = IF STRING 'intensity' SAY ELSE STRING 'doppler velocity' SAY
THEN CRLF STRING 'Change it ?' SAY CIN PUSH 131 = IF 131 TO MEM @W 0 =
IF 1 ELSE 0 THEN MEM ! SORT 0MAP ELSE 77 TO THEN CRLF ;
: VQUERY CRLF STRING 'ARE YOU SURE? ' SAY CIN PUSH
131 = IF 131 TO STRING ' O.K. YOU SAID IT !!! ' SAY ELSE 15 TO
                STRING ' I HAVE ABORTED YOUR REQUEST !! ' SAY POPA 0
THEN ;
: NEW1 STRING 'Enter the new operating parameters as indicated ' SAY CRLF ;
: NINT STRING 'New integration time ' SAY USER DUP 0 = IF VQUERY ELSE
DUP 7 ) IF VQUERY THEN THEN INTTIM MSAVE ;
: CHANGE NEW1 NINT CRLF ;
( ************************************************ )
( * THE REAL HUBBLE BUBBLE WILL STAND UP !!!!!!  * )
( ************************************************ )
: COM 20 + POP NIO 20 SR DOA 0 20 NIO 20 CL ;
: ADX POP NIO 20 SR DOA 0 20 NIO 20 CL ;
: DAT POP DOA 0 20 ;
: STAT NIO 20 SR DIA 0 20 NIO 20 CL DIC 0 20 PUSH ;
: READ DIA 0 20 NIO 20 DIC 0 20 PUSH ;
: MSTAT 200 1 DO STAT . SPACE LOOP ;
: Z 50 1 DO I DAT LOOP ; : W 50 1 DO READ . SPACE LOOP ;
: E 50 1 DO 377 DAT LOOP ;
: INIT 13 ADX 1 DAT 20 DAT 50 DAT 0 DAT 0 DAT ;
: PR 4770 DUP 100 + SWAP DO I @W . SPACE LOOP ;
: PL 13 ADX 1 DAT 20 DAT 110 DAT DAT 0 DAT ;
: *1PAGE 13 ADX 1 DAT 0 ADX ;
: W-PAGE 77 0 DO BYTE PUSH 6017 I + ! LOOP *1PAGE WPAGE ;
: ALL-MEM 50 GT ! 1777 0 DO W-PAGE LOOP ;
: PUSER 40 PREFIX ! BUFFER WORD 1 LAST 2@ = IF 0 LAST !
 -1 ELSE NUMBER THEN 40 PREFIX ! ;
```

```
: DUMP 34777 34000 DO I @W 0. SPACE LOOP ;
: TRUN 10000 1 DO I , I NWRITE *W TBUSY? 1 BWD TBUSY? *R TBUSY? I NREAD LOOP ;
: LOOK 10000 1 DO 0 ERROR ! *R TBUSY? I NREAD ERROR @W 0 = IF ELSE
      STRING 'TAPE ERROR(S) ' SAY ERROR @W 0. SPACE STRING 'BLOCK NO. '
      SAY I 0. CRLF THEN LOOP ;
( ************************************** )
( * REAL TIME CLOCK DEVICE 40          * )
( ************************************** )
: CREAD 1 POP DOC 0 40 SR ; ( READ MODE )
: CWRITE 0 POP DOC 0 40 SR ; ( WRITE MODE )
: RADDX CREAD POP DOA 0 40 PU ISKP 40 BZ JMP -1 1 DIB 0 40 PUSH ; ( READS CLOCK
: WADDX CWRITE POP DOA 0 40 POP DOB 0 40 PU ISKP 40 BZ JMP -1 1 ; ( SETS CLOCK )
: CLKON 1 16 WADDX ; ( 1 @ ADDRESS 16 STARTS IT )
: USER 40 PREFIX ! BUFFER WORD NUMBER CRLF 76 PREFIX ! ;
: MIN STRING 'Mins ' SAY ;
: HOUR STRING 'Hours ' SAY ;
: DAY STRING 'Days ' SAY ;
: MONTH STRING 'Months ' SAY ;
: UNDO DUP 12 SWAP / SWAP OVER 12 * - ;
: STOD BASE DEC MIN USER UNDO 4 WADDX 5 WADDX
                HOUR USER UNDO 6 WADDX 7 WADDX
                 DAY USER UNDO 10 WADDX 11 WADDX
                MONTH USER UNDO 13 WADDX 14 WADDX
                0 2 WADDX 0 3 WADDX
   CRLF RDX ! 43 PREFIX ! ;
: 17WAIT 7777 1 DO 2 RADDX 17 = IF ABORT THEN LOOP STRING 'CLOCK FAIL' SAY ;
: GETALL 17WAIT 2 RADDX
         3 RADDX 4 RADDX 5 RADDX 6 RADDX 7 RADDX
         10 RADDX 11 RADDX 13 RADDX 14 RADDX ;
: GTOD BASE DEC GETALL 12 * + . SPACE SPACE
  12 * + . SPACE SPACE 12 * + . SPACE SPACE
  12 * + . SPACE SPACE 12 * + . RDX ! CRLF ;
( ************************************** )
( * DATA ACQUISITION PROGRAMS......... * )
( ************************************** )
: GDAT 177400 POP DOA 0 43 DOB 0 43 DOC 0 43 DOC 0 41
       50000  POP DOB 0 41 SR ;
: DDAT 50400 50000 DO I @W 0. SPACE LOOP ;
: CDAT 50400 50000 DO 0 I ! LOOP ;
( ************************************** )
( * TEKTRONIX DISPAY SOFTYWARE          )
( ************************************** )
: X POP DOA 0 42 ;
: Y POP DOB 0 42 ;
```

```
: CHECK DUP -1 = IF DROP DROP ELSE SWAP ! THEN ;
: MINIMUM 1 + DUP @W STRING %MINIMUM = % SAY . PUSER CHECK ;
: MAXIMUM DUP @W STRING %MAXIMUM = % SAY . PUSER CHECK ;
: WUSER 40 PREFIX ! BUFFER WORD 1 LAST @W = IF 0 LAST !
 -1 ELSE FIND DUP 0 = IF
 STRING %NOT FOUND% SAY CRLF DROP -1 ELSE 5 + THEN THEN 43 PREFIX ! ;
: BACKFIND 5 - DUP @W 377 LAND STRING %LENGTH = % SAY . SPACE SPACE
 DUP 1 + @W DUP POP MOV 0 0 S COUT TO 2 + @W DUP POP MOV 0 0 S COUT TO ;
: 1ACT STRING %ON MINIMUM  % SAY 3 + DUP @W BACKFIND WUSER CHECK ;
: 2ACT STRING %ON MAXIMUM  % SAY 2 + DUP @W BACKFIND WUSER CHECK ;
: LIMITS 4 * DBASE @W 1 + + DUP DUP DUP MINIMUM MAXIMUM 1ACT 2ACT ;
: VOLTS DBLOCK 37 0 DO I DUP DUP DUP STRING %CHANNEL % SAY
  . POP MOV 0 0 R PUSH DVALS 1 + + @W STRING %  PRESENT VALUE %
  SAY SWAP 1 LAND
  0 = IF POP MOV 0 0 S PUSH ELSE THEN 377 LAND . 4 * DBASE @W 1
  + + DUP 1 + @W
  STRING %  MINIMUM % SAY . STRING %  MAXIMUM %
  SAY @W .  CRLF LOOP ;
: NEWPULSE 40400 40000 DO I DUP DUP STRING %LOCATION % SAY .
  @W STRING %  VALUE % SAY . 40 PREFIX ! SPACE BUFFER WORD
  43 PREFIX ! LAST @W 1 = IF DROP 0 LAST ! ELSE NUMBER DUP
  0 )= IF SWAP ! ELSE DROP DROP ABORT THEN THEN LOOP CRLF ;
: PXFER -400 NWDS ! 40000 MADX ! 137777 ADX3 ! XFER
  100004 MICRO ! -1 NWDS ! MICRO DUP
  MADX ! 1 - 100000 + ADX3 ! XFER ;
: TXOFF 100040 MICRO ! -1 NWDS ! MICRO DUP MADX ! 1 - 100000
  + ADX3 ! XFER ;
: TXON -1 NWDS ! 100010 MICRO ! MICRO MADX ! MICRO 1 - 100000 + ADX3 ! XFER ;
: FAST 2 INTTIM MSAVE STRING % INTEGRATION TIME NOW 20 SECONDS % SAY CRLF
 STRING % MAIN PROGRAM EXECUTING % SAY CRLF MAIN ;
: SLOW 6 INTTIM MSAVE STRING % INTEGRATION TIME NOW 60 SECONDS % SAY CRLF
 STRING % MAIN PROGRAM EXECUTING % SAY CRLF MAIN ;
: START TXON STRING % SYSTEM RESTARTED % SAY CRLF MAIN ;
: MTOD TIMDMA DEC DVALS 17 + DVALS 14 + DO I @W . SPACE LOOP CRLF OCT ;
SP TABLE @W DP @W - DP @W
: MSG STRING 'Welcome to S)A)B)R)E friend !!! ' CRLF SAY
        STRING 'Dictionary pointer = ' CRLF SAY 0.
        STRING 'Free space         = ' CRLF SAY 0.
        STRING 'Top of memory      = ' CRLF SAY 0. ;
MSG FORGET MSG
RESTART
End of Listing
```

Appendix 3

Nova 3 System Software

```
C FILE TRANSFER PROGRAM
C
C THE FOLLOWING ASSEMBLER ROUTINES ARE CALLED
C
C  CALL SETUP(IER)......IER=ERROR CODE
C  CALL READL(ITAR,IER) ITAR=TRANSMIT ARRAY
C  CALL TXMT(CONTROL,IER,ITAR) CONTROL=BLOCK TYPE .TRUE.=)CONTROL
C  CALL RXMT(ILEN,IER,IRAR) ILEN=RECEIVED BLOCK LENGTH
C
        COMPILER NOSTACK
        DIMENSION ITAR(256),IRAR(256)
        DIMENSION JARR(12)
        LOGICAL CONTROL,IFIN,LAST
C
C
      IFIN=.FALSE.
1       TYPE "Are you in originate or answer mode ?"
        TYPE
        ACCEPT "1=Originate  0=Answer  -)",IORIG
        IF (IORIG.GT.1.OR.IORIG.LT.0) GOTO 1
        IF (IORIG.EQ.0) GOTO 11
C
C FILE HANDLING PHASE FIRST
C
        TYPE
        TYPE
        TYPE "What is the RDOS file you wish to transfer ?"
        WRITE(10,1000)
1000    FORMAT(1H ,"-)",Z)
        READ(11,2000)(JARR(I),I=1,12)
2000    FORMAT(12A2)
        CALL OPEN(1,JARR,3,IER,512)
        IF (IER.EQ.1) GOTO 3
        WRITE (10,3000)JARR(1),IER
3000    FORMAT(1H ,"Tried to open file ",S20," But failed with error code ",I4
        STOP
C
C RECEIVE FILE HANDLER
C
11      TYPE
        TYPE "Where do you wish your file to go ?"
        WRITE(10,1000)
        READ(11,2000)(JARR(I),I=1,12)
        CALL OPEN(2,JARR,3,IER,512)
        IF(IER.EQ.1)GOTO 3
        WRITE(10,3000)JARR(1),IER
        STOP
C
C SETUP CONTROL CHARACTERS IN TRANSMIT ARRAY
C
C BLOCK LOOKS LIKE SYN.SYN.SYN.SYN.DLE.STX.....DLE.ETX.
C
3       ITAR(1)=13026K
        ITAR(2)=13026K
        ITAR(3)=10002K
C
C SETUP THE ULM
C
        CALL SETUP(IER)
        TYPE IER
        IF (IER.EQ.0) GOTO 42
        TYPE
        TYPE "Setup error..failed to IDEF ULM error code ",IER
        TYPE "Fatal error causes program HALT"
        STOP
C
```

```
C FIRST OF ALL SEND AN ENQ BLOCK
C
42      IF(IORIG.EQ.0)GOTO 21
        ICOUNT=0
        TYPE "CALL TXMT"
41      CALL TENQ(ITAR)
        TYPE "CALL RXMT"
        CALL RXMT(ILEN,IER,IRAR)
        IF (IER.EQ.2) GOTO 4
        ICOUNT=ICOUNT+1
        IF(ICOUNT.LE.6) GOTO 41
        TYPE
        TYPE "DIFFICULTY..Call error...HALTING"
        STOP
C
C OK SO NOW READ THE FILE
C
4       CALL READL(ITAR,IER)
        IF (IER.EQ.6) IFIN=.TRUE.
        IF (IER.EQ.6.OR.IER.EQ.0) GOTO 5
        TYPE
        TYPE "Fatal READ on file error code ",IER
        TYPE "FATAL....STOP"
        STOP
C
C AND TRANSMIT IT
C
5       CALL TDUM
        CALL TDUM
        CALL TDUM
        CALL TDUM
        CONTROL=.FALSE.
        CALL TXMT(CONTROL,IER,ITAR)
C
C NOW GET THE ACKNOWLEDGMENT
C
8       CALL RXMT(ILEN,IER,IRAR)
        TYPE "STATE 5 CODE=",IER
C
C CHECK THE RESPONSE
C
        IF(IER.EQ.2.AND.IFIN) GOTO 10
        IF(IER.EQ.2.AND..NOT.IFIN) GOTO 4
        IF(IER.EQ.3) GOTO 5
        IF(IER.EQ.4) GOTO 5
        IF(IER.EQ.5.OR.IER.EQ.6.OR.IER.EQ.7) GOTO 9
C
C SOME UNDEFINED ERROR HERE
C
        TYPE "Fatal RECEIVE error code ",IER
        TYPE "PROGRAM HALT"
        STOP
C
C WE GOT GARBAGE BACK
C
9       ICOUNT=0
        CALL TENQ(ITAR)
        GOTO 8
C
C END OF FILE FOUND SO WE CAN TURN LINE AROUND
C
10      ICOUNT=0
14      CALL TEOB(ITAR)
        CALL RXMT(ILEN,IER,IRAR)
        IF(IER.EQ.2)GOTO 12
        ICOUNT=ICOUNT+1
        IF(ICOUNT.LE.6)GOTO 14
        TYPE
        TYPE "Failed END OF BLOCK"
        STOP
C
```

```
C NOW DECIDE WHAT TO DO
C
12      TYPE "FINISHED OK"
        STOP
C
C*************************************************************
C* THE ANSWER PART OF THE TRANSFER PROGRAM ACCEPTS DATA      *
C*************************************************************
C
C WAIT UNTIL RING
C
21      CONTINUE
        LAST=.TRUE.
C
C NOW GET RECEIVE DATA
C
        ICOUNT=0
23      CALL RXMT(ILEN,IER,IRAR)
        TYPE "23.IER=",IER
        IF(IER.EQ.4) GOTO 24
        CALL TENQ(ITAR)
        ICOUNT=ICOUNT+1
        IF(ICOUNT.LE.6) GOTO 23
        TYPE "Difficulty in establishing call"
        TYPE "HALTING"
        STOP
C
C GOT AN ENQ FROM OTHER END SO SEND ACK
C
24      CALL TACK(ITAR)
C
C NOW SHOULD GET DATA
C
25      CALL RXMT(ILEN,IER,IRAR)
        TYPE "25.IER=",IER
C
C GET THE NEW RESPONSE
C
        IF (IER.EQ.4) GOTO 31
        IF (IER.EQ.2.OR.IER.EQ.3) GOTO 26
        IF (IER.EQ.6.OR.IER.EQ.7) GOTO 26
        IF (IER.EQ.1) GOTO 27
        IF (IER.EQ.0) GOTO 28
        IF (IER.EQ.5) GOTO 30
C
C SOMETHING WRONG HERE
C
        TYPE "Something is wrong here error code ",IER
        TYPE "HALTING"
        STOP
C
C WE TIMED OUT
C
26      CALL TENQ(ITAR)
        GOTO 25
C
C WE GOT ERRORS
C
27      CALL TNAK(ITAR)
        LAST=.FALSE.
        GOTO 25
C
```

```
C WE ARE ALL O.K.
C
28      ILEN=ILEN-1
        WRITE BINARY(2)(IRAR(I),I=1,ILEN)
        CALL TACK(ITAR)
        LAST=.TRUE.
        GOTO 25
C
C TRANSMIT LAST RESPONSE
C
31      IF(LAST) CALL TACK(ITAR)
        IF(.NOT.LAST) CALL TNAK(ITAR)
        GOTO 25
C
C FINALLY THE END OF BLOCK
C
30      CALL TACK(ITAR)
        TYPE
        TYPE "DONE O.K."
        STOP
        END
```

```
0001  FTXMT
                        .TITLE FTXMT
                        .ENT TXMT
                        .EXTD .CPYL,.FRET
                        .EXTN DATL RFLG TFLG MFLG TDAT RDAT STAT CODE
        000001          .TXTM 1
        000034          .DUSR ULM=34
                        .NREL
 00000'000050 S0
 00001'006001$TXMT:     JSR @.CPYL          ; GET EM
 00002'054546           STA 3,SAV3          ; SAVE IT BOY
                ;       NIOS ULM            ; INITIALISE IT
                ;       SKPBZ ULM           ; WAIT
                ;       JMP .-1             ; NOW
 00003'020556           LDA 0,RTSON         ; RTS='1'
 00004'062034           DOB 0,ULM           ; SET
 00005'020545           LDA 0,SYN           ; GET A SYNC CHAR
 00006'042540           STA 0,@.TDAT        ; STORE IT
 00007'020542           LDA 0,SL            ; SET LINE
 00010'061034           DOA 0,ULM           ; DONE
 00011'102520           SUBZL 0,0           ; +1
 00012'063034           DOC 0,ULM           ; LINE=ON
 00013'004521           JSR TCHAR+1         ; WAIT TILL SYNC CHAR SENT
 00014'034534           LDA 3,SAV3          ; RESTORE ACC 3
 00015'031613           LDA 2,-165,3        ; ADDRESS OF ARRAY O.K.
 00016'027611           LDA 1,@-167,3       ; GET CONTROL
 00017'125005           MOV 1,1,SNR         ; CONTROL/DATA ?
 00020'000420           JMP DATAB           ; DATA BLOK
 00021'024533           LDA 1,CONL          ; CONTROL BLOCK LENGTH
 00022'124400           NEG 1,1             ; 2'S
                        ;
                        ;
 00023'021000 NEXT:     LDA 0,0,2           ; GET A WORD
 00024'034531           LDA 3,MASK          ; 377
 00025'101300           MOVS 0,0            ; GET HI BYTE
 00026'163420           ANDZ 3,0            ; ISOLATED
 00027'004504           JSR TCHAR           ; OUTPUT THE BYTE
 00030'034525           LDA 3,MASK          ; 377
 00031'021000           LDA 0,0,2           ; GET WORD
 00032'163420           ANDZ 3,0            ; LOW BYTE
 00033'004500           JSR TCHAR           ; OUTPUT IT
 00034'151400           INC 2,2             ; UP POINTER
 00035'125424           INCZ 1,1,SZR        ; DONE YET ?
 00036'000765           JMP NEXT            ; SEND NEXT
 00037'000463           JMP FINISH          ; YES SO GO
                        ;
                        ; DATA BLOCK FORMAT
                        ;
 00040'024520 DATAB:    LDA 1,M3            ; THREE WORDS (-3)
                        ;
                        ; FIRST THREE WORDS TRANSPARENTLY
                        ;
 00041'021000 NEXT0:    LDA 0,0,2           ; GET WORD
 00042'034513           LDA 3,MASK          ; 377
 00043'101300           MOVS 0,0            ; TOP BYTE
 00044'163420           ANDZ 3,0            ; GET BYTE
 00045'004466           JSR TCHAR           ; SEND IT
 00046'034507           LDA 3,MASK          ; 377
 00047'021000           LDA 0,0,2           ; GET WORD
 00050'163420           ANDZ 3,0            ; GET BOTTOM BYTE
0002  FTXMT
```

```
00051'004462            JSR TCHAR       ; OUTPUT IT
00052'151400            INC 2,2         ; UP POINTER
00053'125404            INC 1,1,SZR     ; DONE 3 ?
00054'000765            JMP NEXT0       ; NOT YET
                        ;
                        ; NOW THE DATA PART OF THE BLOCK
                        ;
00055'026472 TRAIL:     LDA 1,@.DATL    ; DATA LENGTH
00056'125220            MOVZR 1,1       ; WORD POINTER
00057'125400            INC 1,1         ; +1 FOR CRC
00060'124400            NEG 1,1         ; 0.K.
00061'021000 NEXT1:     LDA 0,0,2       ; GET A WORD
00062'034473            LDA 3,MASK      ; 377
00063'101300            MOVS 0,0        ; GET HI BYTE
00064'163420            ANDZ 3,0        ; ISOLATE IT
00065'034466            LDA 3,DLE       ; DLE ?
00066'162435            SUBZ# 3,0,SNR   ; WELL
00067'004444            JSR TCHAR       ; SEND IT
00070'004443            JSR TCHAR       ; SEND IT
00071'021000            LDA 0,0,2       ; GET WORD
00072'034463            LDA 3,MASK      ; 377
00073'163420            ANDZ 3,0        ; ISOLATE IT
00074'034457            LDA 3,DLE       ; DLE ?
00075'162435            SUBZ# 3,0,SNR   ; WELL ?
00076'004435            JSR TCHAR       ; PUT 2 IN
00077'004434            JSR TCHAR       ; SEND IT
00100'151400            INC 2,2         ; BUMP POINTER
00101'125424            INCZ 1,1,SZR    ; DONE YET ?
00102'000757            JMP NEXT1       ; NEXT
                        ;
                        ; FINALLY THE TRAILER
                        ;
00103'126520            SUBZL 1,1       ; GENERATE +1
00104'125400            INC 1,1
00105'124020            COMZ 1,1        ; 177776
00106'021000 NEXT2:     LDA 0,0,2       ; GET WORD
00107'034446            LDA 3,MASK      ; 377
00110'101300            MOVS 0,0        ; TOP
00111'163420            ANDZ 3,0        ; GET IT
00112'004421            JSR TCHAR       ; ZAP
00113'034442            LDA 3,MASK      ; 377
00114'021000            LDA 0,0,2
00115'163420            ANDZ 3,0        ; GET BYTE
00116'004415            JSR TCHAR       ; OK
00117'151400            INC 2,2         ; BUMP POINTER
00120'125424            INCZ 1,1,SZR    ; DONE ?
00121'000765            JMP NEXT2       ; NO !
                        ;
                        ; DONE NOW
                        ;
00122'022423 FINISH:    LDA 0,@.TFLG
00123'101005            MOV 0,0,SNR
00124'000776            JMP FINISH
00125'020424            LDA 0,SL        ; SET LINE
00126'061034            DOA 0,ULM       ; SET IT
00127'102400            SUB 0,0         ; +0
00130'063034            DOC 0,ULM       ; TURN SECTION OFF
00131'034417            LDA 3,SAV3      ; GET 3 BACK
00132'006002$           JSR @.FRET
0003  FTXMT
```

```
                                ;
                                ; BASIC TX ROUTINE
                                ;
00133'042413 TCHAR:   STA 0,@.TDAT            ; STORE CHAR
00134'054410          STA 3,SAV33             ; SAVE 3
00135'036410          LDA 3,@.TFLG            ; GONE YET ?
00136'175005          MOV 3,3,SNR             ; ?     .
00137'000776          JMP .-2                 ; NO SO WAIT
00140'176420          SUBZ 3,3                ; CLEAR FLAG
00141'056404          STA 3,@.TFLG            ; STORE IT
00142'034402          LDA 3,SAV33             ; RETURN
00143'001400          JMP 0,3          ; NO
                                ;
                                ; DATA REQUIRED
                                ;
00144'000000 SAV33:   0
00145'077777 .TFLG:   TFLG
00146'077777 .TDAT:   TDAT
00147'077777 .DATL:   DATL
00150'000000 SAV3:    0
00151'000021 SL:      21
00152'000026 SYN:     26
00153'000020 DLE:     20
00154'000012 CONL:    12
00155'000377 MASK:    377
00156'000000 TEMP:    0
00157'000012 TEN:     12
00160'177775 M3:      -3
00161'100003 RTSON:   100003
                      .END
  0004  FTXMT
CODE    077777 X
CONL    000154'
DATAB   000040'
DATL    000147'X
DLE     000153'
FINIS   000122'
M3      000160'
MASK    000155'
MFLG    077777 X
NEXT    000023'
NEXT0   000041'
NEXT1   000061'
NEXT2   000106'
RDAT    077777 X
RFLG    077777 X
RTSON   000161'
SAV3    000150'
SAV33   000144'
SL      000151'
STAT    077777 X
SYN     000152'
TCHAR   000133'
TDAT    000146'X
TEMP    000156'
TEN     000157'
TFLG    000145'X
TRAIL   000055'
TXMT    000001'
.CPYL   000001$X
.DATL   000147'
.FRET   000002$X
.TDAT   000146'
.TFLG   000145'
End of Listing
```

```
0001  FRXMT
                        .TITLE FRXMT
                        .ENT RXMT
                        .EXTD .CPYL,.FRET
                        .EXTD CRC
                        .EXTN RFLG TFLG MFLG TDAT RDAT STAT
                        .EXTN CODE DATL DARAY
        000001          .TXTM 1
        000034          .DUSR ULM=34
                        .NREL
 00000'000050 50
 00001'000000 SAV3:     0
 00002'077777 .DARAY:   DARAY
                        ;
                        ; RECEIVE PROGRAM ERROR RETURNS ARE :-
                        ;
                        ; 0=DATA BLOCK WITH NO ERRORS
                        ; 1=DATA BLOCK WITH BURST ERROR
                        ; 2=CONTROL BLOCK ACK
                        ; 3=CONTROL BLOCK NAK
                        ; 4=CONTROL BLOCK ENQ
                        ; 5=SYNC CHARACTERS
                        ; 6=UNDEFINED CONTROL SEQUENCE
                        ; 7=TIMEOUT
                        ;
                        ;
 00003'006001$RXMT:     JSR @.CPYL          ; FORTRAN .. YEUGH
 00004'054775           STA 3,SAV3          ; SAVE ACC 3
                ;       NIOS ULM            ; INIT ULM
                ;       SKPBZ ULM           ; WAIT TILL
                ;       JMP .-1             ; DONE
 00005'020504           LDA 0,RTSOFF        ; RTS='0'
 00006'062034           DOB 0,ULM           ; SET IT AND ULM=ON
 00007'031613           LDA 2,-165,3        ; GET ARRAY ADDRESS
 00010'052772           STA 2,@.DARAY       ; STORE IT FOR CRC
 00011'020477           LDA 0,SL            ; SYNC LINE
 00012'061034           DOA 0,ULM           ; SET IT
 00013'102520           SUBZL 0,0           ; +1
 00014'063034           DOC 0,ULM           ; LINE IS ON
 00015'126420           SUBZ 1,1            ; GENERATE +0
 00016'044466           STA 1,STAT1         ; STATUS
 00017'044463           STA 1,START         ; START=0
 00020'044463           STA 1,ERROR         ; ERROR=0
 00021'044464           STA 1,END           ; END  =0
 00022'044465           STA 1,LASTD         ; LASTD=0
 00023'044456           STA 1,TEMP0         ; COUNTER=0
                        ;
 00024'004467 NEXT:     JSR RCHAR           ; GET A CHAR
 00025'101300           MOVS 0,0            ; TOP BYTE
 00026'041000           STA 0,0,2           ; STORE IT
 00027'004464           JSR RCHAR           ; GET A CHAR
 00030'025000           LDA 1,0,2           ; GET LAST VALUE
 00031'123020           ADDZ 1,0            ; ADD IN NEW
 00032'041000           STA 0,0,2           ; STORE IT
 00033'151400           INC 2,2             ; BUMP POINTER
 00034'024445           LDA 1,TEMP0         ; CURRENT NO.
 00035'020451           LDA 0,MAX           ; MAX ALLOWED
 00036'106533           SUBZL# 0,1,SNC      ; FULL ?
 00037'000404           JMP FINISH
 00040'024445           LDA 1,END           ; DONE ?
0002  FRXMT
```

```
00041'125025          MOVZ 1,1,SNR     ; WELL ?
00042'000762          JMP NEXT         ; LOOP
00043'020445 FINISH:  LDA 0,SL         ; SYNC LINE
00044'061034          DOA 0,ULM        ; SET IT
00045'102420          SUBZ 0,0         ; +0
00046'063034          DOC 0,ULM        ; SECTION = OFF
                      ;
00047'020435          LDA 0,STAT1      ; GET STATUS
00050'101024          MOVZ 0,0,SZR     ; SHOULD BE ZERO
00051'000421          JMP FIN1         ; GOT-A-CONTROL BLOCK (FORGET CRC)
                      ;
00052'126520          SUBZL 1,1        ; +1
00053'020426          LDA 0,TEMP0      ; NO OF WORDS GOT
00054'101025          MOVZ 0,0,SNR     ; NO. OF WORDS=0 ?
00055'000415          JMP FIN1         ; YES !
00056'122420          SUBZ 1,0         ; NO.-1
00057'122420          SUBZ 1,0         ; NO. OF WORDS -1 (BYTES-2)
00060'042432          STA 0,@.DATL     ; STORE IT FOR CRC
00061'101005          MOV 0,0,SNR      ; NO OF BYTES=0
00062'000410          JMP FIN1         ; NO.=0
00063'126420          SUBZ 1,1         ; ZERO 1 FOR CRC CALC
00064'006575          JSR @.CRC        ; DO IT
00065'022573          LDA 0,@.CODE     ; GET RESULT
00066'024416          LDA 1,STAT1      ; STATUS
00067'101024          MOVZ 0,0,SZR     ; CODE=0 ?
00070'024561          LDA 1,A1         ; NO SO CRC ERROR
00071'044413          STA 1,STAT1      ; RESTORE STATUS
                      ;
00072'024412 FIN1:    LDA 1,STAT1      ; GET STATUS
00073'034706          LDA 3,SAV3       ; GET ACC 3
00074'047612          STA 1,@-166,3    ; STORE STATUS
00075'026415          LDA 1,@.DATL     ; NO. OF BYTES
00076'125220          MOVZR 1,1        ; NO. OF WORDS
00077'047611          STA 1,@-167,3    ; STORE IT
00100'006002$         JSR @.FRET       ; RETURN TO FORTRAN
                      ;
                      ; SOME DATA
                      ;
00101'000000 TEMP0:   0
00102'000000 START:   0
00103'000000 ERROR:   0
00104'000000 STAT1:   0
00105'000000 END:     0
00106'000764 MAX:     500.
00107'000000 LASTD:   0
00110'000020 SL:      20
00111'100001 RTSOFF:  100001
00112'077777 .DATL:   DATL
                      ;
                      ; BASIC RX ROUTINE
                      ;
00113'054532 RCHAR:   STA 3,SAV33
00114'176420          SUBZ 3,3         ; GENERATE +0
00115'054531          STA 3,TEMP       ; TIMER
00116'034544          LDA 3,DEL1       ; DELAY=200
00117'054530          STA 3,TEMP1      ; TIMER
00120'004502          JSR TIME1        ; TIMEOUT
00121'176420          SUBZ 3,3         ; ZERO FLAG
00122'056521          STA 3,@.RFLG     ; CLEAR FLAG
0003  FRXMT
```

```
00123'022521           LDA 0,@.RDAT     ; GET DATA
00124'034516           LDA 3,DLE        ; DLE
00125'162434           SUBZ# 3,0,SZR    ; IS IT ?
00126'000406           JMP NODLE        ; NO !
00127'034760           LDA 3,LASTD      ; LAST ONE A DLE ?
00130'175004           MOV 3,3,SZR      ; WELL ?
00131'000466           JMP ONDLE        ; YES !
00132'010755           ISZ LASTD        ; NO SO SET IT
00133'000460           JMP EXIT1        ; LEAVE
                       ;
                       ;
00134'034753 NODLE:    LDA 3,LASTD      ; WAS LAST ONE A DLE
00135'175025           MOVZ 3,3,SNR     ; WELL ?
00136'000455           JMP EXIT1        ; WAIT FOR IT !
00137'176420           SUBZ 3,3         ; +0
00140'054747           STA 3,LASTD      ; CLEAR DLE FLAG
00141'024473           LDA 1,STX        ; START OF TEXT ?
00142'122435           SUBZ# 1,0,SNR    ; ?
00143'000423           JMP STX1         ; YES !
00144'024471           LDA 1,ETX        ; END OF TEXT ?
00145'122435           SUBZ# 1,0,SNR    ; ?
00146'000423           JMP ETX1         ; YES
00147'024467           LDA 1,SYN        ; SYNC
00150'122435           SUBZ# 1,0,SNR    ; OK
00151'000423           JMP SYN1         ; ZAP
00152'024465           LDA 1,ACK        ; ACK
00153'122435           SUBZ# 1,0,SNR    ; WELL
00154'000423           JMP ACK1                 ; ACK
00155'024463           LDA 1,NAK        ; NAK ?
00156'122435           SUBZ# 1,0,SNR    ; WELL
00157'000423           JMP NAK1         ; OK
00160'024461           LDA 1,ENQ        ; ENQ?
00161'122435           SUBZ# 1,0,SNR    ; WELL
00162'000423           JMP ENQ1         ; YES
                       ;
                       ; ERROR UNDEFINED CONTROL CHARS
                       ;
00163'024473           LDA 1,A6         ; ERROR CODE 6
00164'044720           STA 1,STAT1      ; STORE IT
00165'000423           JMP EXIT         ; GONE
                       ;
                       ; CONTROL CHARACTER SECTION
                       ;
00166'010714 STX1:     ISZ START        ; START COLLECTING CHARACTERS
00167'101000           MOV 0,0
00170'000724           JMP RCHAR+1      ; GET ANOTHER
                       ;
00171'010714 ETX1:     ISZ END          ; FINISH
00172'101000           MOV 0,0
00173'000415           JMP EXIT         ; OK
                       ;
00174'024461 SYN1:     LDA 1,A5         ; ERROR CODE 5
00175'044707           STA 1,STAT1      ; O.K.
00176'000412           JMP EXIT         ; LEAVE
                       ;
00177'024453 ACK1:     LDA 1,A2         ; ERROR CODE 2
00200'044704           STA 1,STAT1      ; O.K.
00201'000407           JMP EXIT         ; LEAVE
                       ;
0004  FRXMT
```

```
00202'024451 NAK1:   LDA 1,A3          ; ERROR CODE 3
00203'044701         STA 1,STAT1       ; O.K.
00204'000404         JMP EXIT          ; LEAVE
                     ;
00205'024447 ENQ1:   LDA 1,A4          ; ERROR CODE 4
00206'044676         STA 1,STAT1       ; O.K.
00207'000401         JMP EXIT          ; LEAVE
                     ;
00210'034435 EXIT:   LDA 3,SAV33       ; GET 3 BAK
00211'010670         ISZ TEMP0         ; COUNTER
00212'001400         JMP 0,3           ; LEAVE
                     ;
00213'024667 EXIT1:  LDA 1,START       ; HAVE WE STARTED YET ?
00214'125024         MOVZ 1,1,SZR      ; WELL ?
00215'000773         JMP EXIT          ; YES SO LEAVE
00216'000676         JMP RCHAR+1       ; NO SO GET ANOTHER
                     ;
00217'176420 ONDLE:  SUBZ 3,3          ; CLEAR DLE FLAG
00220'054667         STA 3,LASTD       ; DONE
00221'000673         JMP RCHAR+1       ; GET ANOTHER CHARACTER
                     ;
                     ; TEST AND TIME ROUTINE
                     ;
00222'026421 TIME1:  LDA 1,@.RFLG      ; GET THE FLAG
00223'125004         MOV 1,1,SZR       ; IS IT 0
00224'001400         JMP 0,3           ; NO SO NORMAL RETURN
00225'014421         DSZ TEMP          ; TIMER
00226'000774         JMP TIME1         ; O.K.
00227'014420         DSZ TEMP1         ; TIMER
00230'000772         JMP TIME1         ; O.K.
00231'024426         LDA 1,A7          ; ERROR CODE 7
00232'044652         STA 1,STAT1       ; STATUS=TIMEOUT
00233'000610         JMP FINISH        ; EXIT
                     ;
                     ; DATA
                     ;
00234'000002 STX:    2
00235'000003 ETX:    3
00236'000026 SYN:    26
00237'000006 ACK:    6
00240'000025 NAK:    25
00241'000005 ENQ:    5
00242'000020 DLE:    20
00243'077777 .RFLG:  RFLG
00244'077777 .RDAT:  RDAT
00245'000000 SAV33:  0
00246'000000 TEMP:   0
00247'000000 TEMP1:  0
00250'000005 FIVE:   5
00251'000001 A1:     1
00252'000002 A2:     2
00253'000003 A3:     3
00254'000004 A4:     4
00255'000005 A5:     5
00256'000006 A6:     6
00257'000007 A7:     7
00260'077777 .CODE:  CODE
00261'000003$.CRC:   CRC
00262'000040 DEL1:   40
0005  FRXMT
                     .END
```

```
0001  TREAD
                          ;
                          ; CALLED BY READL(DATA ARRAY, WORK ARRAY, ERROR)
                          ;
                          .TITLE TREADL
                          .ENT DATL
                          .ENT READL
                          .EXTN CODE
                          .ENT WORK DARAY
                          .EXTD .CPYL,.FRET
                          .EXTD CRC
        000001            .TXTM 1
                          .NREL
 00000'000150 150
 00001'006001$READL:      JSR @.CPYL
 00002'021611             LDA 0,-167,3     ; THE ARRAY ADDX
 00003'040447             STA 0,DARAY      ; FOR CRC
 00004'024436             LDA 1,THR        ; GET +3
 00005'123020             ADDZ 1,0         ; FIRST FREE LOCATION IN ARRAY
 00006'054433             STA 3,SAV3       ; SAVE ACC3
 00007'101120             MOVZL 0,0        ; GENERATE BYTE POINTER
 00010'024433             LDA 1,NBYTE      ; NO. OF BYTES TO READ
 00011'006017             .SYSTM           ; CALL RDOS
 00012'015002             .RDS 2           ; GET LINE
 00013'101001             MOV 0,0,SKP              ; ERROR ?
                          ;
                          ; IF ERROR THE NEXT INSTRUCTION IS SKIPPED
                          ;
 00014'152420             SUBZ 2,2         ; FORCE IER=0
                          ;
 00015'050432             STA 2,ECODE      ; STORE ERROR CODE FOR LATER
                          ;
 00016'125232             MOVZR# 1,1,SZC   ; ODD NO. OF BYTES ?
 00017'125400             INC 1,1          ; YES SO MAKE EVEN
 00020'044425             STA 1,DATL       ; STORE BYTE COUNT
 00021'101220             MOVZR 0,0        ; GET WORD POINTER
 00022'135220             MOVZR 1,3        ; GET NO. OF WORDS
 00023'117020             ADDZ 0,3         ; NEXT FREE LOCATION
                          ;
 00024'054422             STA 3,NEWL       ; SAVE NEXT FREE LOCATION FOR LATER
 00025'024415             LDA 1,THR        ; OFFSET TO CALC CRC
 00026'006423             JSR @.CRC        ; CALC CRC
 00027'034417             LDA 3,NEWL       ; GET ADDRESS BACK
 00030'022420             LDA 0,@.CODE     ; GET CRC
 00031'041400             STA 0,0,3        ; STORE IT
                          ;
 00032'175400             INC 3,3          ; RESERVE 1 WORD FOR CRC
 00033'020411             LDA 0,ENDIT      ; ETX
 00034'041400             STA 0,0,3        ; STORE ETX
 00035'034404             LDA 3,SAV3       ; GET 3
 00036'030411             LDA 2,ECODE      ; GET ERROR CODE
 00037'053612             STA 2,@-166,3    ; STORE ERROR CODE
 00040'006002$            JSR @.FRET       ; RETURN
                          ;
                          ; DATA
                          ;
 00041'000000 SAV3:       0
 00042'000003 THR:        3
 00043'000620 NBYTE:      400.
 00044'010003 ENDIT:      10003    ; .DLE.ETX.
0002  TREAD
 00045'000000 DATL:       0
 00046'000000 NEWL:       0
 00047'000000 ECODE:      0
 00050'077777 .CODE:      CODE
 00051'000003$.CRC:       CRC
 00052'000000 DARAY:      0
 00053'000054'WORK:       .+1
        000400            .BLK 400
                          .END
```

```
0001  FSETU
                          .TITLE FSETUP
                          .NREL
        000001            .TXTM 1
                          .ENT SETUP IN1
                          .EXTN AIN1
                          .EXTD .CPYL .FRET
 00000'000010 10
        000034            .DUSR ULM=34
 00001'006001$SETUP:      JSR @.CPYL
 00002'054437             STA 3,SAV3              ; SAVE 3
 00003'152400             SUB 2,2                 ; CLEAR ERROR
 00004'053611             STA 2,@-167,3           ; =0
 00005'020443             LDA 0,UDEV              ; DEVICE 34
 00006'024443             LDA 1,IN1               ; DCI
 00007'005017             .SYSTM            ; CALL RDOS
 00010'021007             .IDEF             ; CALL IDEF
 00011'000425             JMP ERROR               ; ERROR ?
 00012'060134 NRET:       NIOS ULM                ; SETUP
 00013'020430             LDA 0,SL                ; LINE NO.
 00014'024430             LDA 1,LC                ; CHARACTERISTICS
 00015'030430             LDA 2,SYN               ; SYNC CHAR
 00016'034430             LDA 3,MCS               ; MODEM CONTROL
 00017'063534             SKPBZ ULM               ; WAIT
 00020'000777             JMP .-1                 ; OK
 00021'061034             DOA 0,ULM
 00022'067034             DOC 1,ULM
 00023'073034             DOC 2,ULM
 00024'076034             DOB 3,ULM               ; DONE
 00025'024422             LDA 1,DLE               ; DATA LINK
 00026'067034             DOC 1,ULM               ; SETIT
 00027'101400             INC 0,0                 ; TX SECTION
 00030'061034             DOA 0,ULM               ; SET SECTION
 00031'073034             DOC 2,ULM               ; SET SYNC
 00032'067034             DOC 1,ULM               ; SET DLE
 00033'060234             NIOC ULM                ; ONLINE
 00034'034405             LDA 3,SAV3              ; GET 3
 00035'006002$            JSR @.FRET              ; RETURN
                          ;
                          ; ERROR ROUTINE
                          ;
 00036'034403 ERROR:      LDA 3,SAV3
 00037'053611             STA 2,@-167,3
 00040'000752             JMP NRET
                          ;
                          ; DATA
                          ;
 00041'000000 SAV3:       0
 00042'000200 MASK:       200
 00043'000020 SL:         20
 00044'101030 LC:         101030
 00045'040026 SYN:        40026
 00046'100001 MCS:        100001
 00047'140020 DLE:        140020
 00050'000034 UDEV:       34
                          ;
                          ; INTERRUPT DEVICE TABLE FOR RDOS
                          ;
 00051'000052'IN1:        .+1
 00052'000000             0
0002  FSETU
 00053'000200             200
 00054'077777             AIN1
                          ;
                          ; END
                          ;
                          .END
```

```
0001  FISER
                          .TITLE FISERV
                          .NREL
                          .EXTN .UIEX
                          .ENT AIN1
                          .ENT TFLG RFLG MFLG TDAT RDAT STAT
       000034             .DUSR ULM=34
 00000'054435 AIN1:       STA 3,SAV3
 00001'050433             STA 2,SAV2
 00002'060434             DIA 0,ULM          ; GET LINE/SECTION
 00003'101222             MOVZR 0,0,SZC      ; TX/RX ?
 00004'000417             JMP TSERV          ; TX
 00005'062434             DIC 0,ULM          ; RX/MODEM ?
 00006'101222             MOVZR 0,0,SZC      ; RX
 00007'000407             JMP MSERV          ; MODEM
 00010'040433             STA 0,STAT         ; SAVE STATUS
 00011'061634             DIBC 0,ULM         ; GET DATA
 00012'040430             STA 0,RDAT         ; STORE IT
 00013'102520             SUBZL 0,0          ; INTERRUPT=1
 00014'040422             STA 0,RFLG         ; OK
 00015'000413             JMP FINISH         ; END
                          ;
                          ;
 00016'040425 MSERV:      STA 0,STAT         ; SAVE STATUS
 00017'102520             SUBZL 0,0          ; +1
 00020'040417             STA 0,MFLG         ; INTERRUPT=1
 00021'060234             NIOC ULM           ; CLEAR INTERRUPT
 00022'000406             JMP FINISH         ; END
                          ;
                          ;
 00023'060234 TSERV:      NIOC ULM           ; CLEAR INTERRUPT
 00024'020415             LDA 0,TDAT         ; GET TX DATA
 00025'062034             DOB 0,ULM          ; TRANSMIT IT
 00026'102520             SUBZL 0,0          ; +1
 00027'040411             STA 0,TFLG         ; INTERRUPT=1
                          ;
                          ;
 00030'030404 FINISH:     LDA 2,SAV2         ; GET 2
 00031'034404             LDA 3,SAV3         ; AND 3
 00032'126400             SUB 1,1            ; FORCE NO RE-SCHEDULING
 00033'077777             .UIEX              ; EXIT
                          ;
                          ; DATA
                          ;
 00034'000000 SAV2:       0
 00035'000000 SAV3:       0
 00036'000000 RFLG:       0
 00037'000000 MFLG:       0
 00040'000000 TFLG:       0
 00041'000000 TDAT:       0
 00042'000000 RDAT:       0
 00043'000000 STAT:       0
                          ;
                          .END
```

```
0001  FCRC
                        ; CRC CHECK SEQUENCE
                        ; CALLED FROM READL
                        ;
                        .TITLE FCRC
                        .ENT CRC CODE
                        .EXTN DATL  DARAY  WORK
                        .EXTD .CPYL,.FRET
        000001          .TXTM 1
                        .NREL
                        ;
                        ; ON ENTRY ACC1=ARRAY OFFSET TO CALC CRC
                        ;
 00000'054501 CRC:      STA 3,SAV3          ; SAVE 3
 00001'032512           LDA 2,@.DARAY       ; GET DATA ARRAY
 00002'133020           ADDZ 1,2            ; START OF DATA
 00003'036511           LDA 3,@.WORK        ; TEMP ARRAY
 00004'026503           LDA 1,@.DATL        ; DATA LENGTH(EVEN NO. BYTES)
 00005'125220           MOVZR 1,1           ; NO. OF WORDS
 00006'124400           NEG 1,1             ; 2'S
 00007'021000 COPY:     LDA 0,0,2           ; GET WORD
 00010'041400           STA 0,0,3           ; STORE IT
 00011'151400           INC 2,2             ; INC COUNTER
 00012'175400           INC 3,3             ; INC COUNTER
 00013'125404           INC 1,1,SZR         ; COUNT
 00014'000773           JMP COPY            ; NEXT WORD
 00015'102420           SUBZ 0,0            ; +0
 00016'041400           STA 0,0,3           ; ZERO LAST WORD
                        ;
                        ; SO FAR SO GOOD
                        ;
 00017'022470           LDA 0,@.DATL        ; NO OF WORD
 00020'101220           MOVZR 0,0           ; NO OF WORDS
 00021'040464           STA 0,NWDS          ; OK
 00022'101120           MOVZL 0,0           ; *2
 00023'101120           MOVZL 0,0           ; *4
 00024'101120           MOVZL 0,0           ; *8
 00025'101120           MOVZL 0,0           ; *16
 00026'040460           STA 0,NBITS         ; NO. OF BITS
                        ;
 00027'036465           LDA 3,@.WORK        ; GET @.WORK ARRAY
 00030'054453           STA 3,ARRAY         ; STORE FOR LATER
 00031'034452 LP1:      LDA 3,ARRAY         ; GET IT BACK
 00032'031400           LDA 2,0,3           ; GET-A-WORD
 00033'151122           MOVZL 2,2,SZC       ; OK
 00034'000406           JMP DVD             ; DIVIDE BY GENERATOR POLY
 00035'051400           STA 2,0,3           ; RESTORE WORD
 00036'004425           JSR SHL             ; LEFT
 00037'014447           DSZ NBITS           ; RUN OUT OF BITS ?
 00040'000771           JMP LP1             ; NO SO CONTINUE
 00041'000415           JMP FINISH          ; END IT ALL
 00042'051400 DVD:      STA 2,0,3           ; RESTORE WORD
 00043'004420           JSR SHL             ; SHIFT LEFT
 00044'034437           LDA 3,ARRAY         ; GET POINTER BACK
 00045'021400           LDA 0,0,3           ; GET WORD
 00046'024444           LDA 1,POLY          ; POLYNOMIAL
 00047'131000           MOV 1,2
 00050'113520           ANDZL 0,2
 00051'107000           ADD 0,1
 00052'146400           SUB 2,1             ; EX-OR DONE
0002  FCRC
```

```
00053'045400          STA 1,0,3        ; REPLACE ARRAY CONTENTS
00054'014432          DSZ NBITS        ; RUN OUT OF BITS ?
00055'000754          JMP LP1          ; SHIFT AGAIN
                      ;
                      ; SO WE HAVE DONE NOW
                      ;
00056'034425 FINISH:  LDA 3,ARRAY      ; GET ARRAY
00057'021400          LDA 0,0,3        ; CRC SEQUENCE
00060'034421          LDA 3,SAV3       ; GET RETURN
00061'040430          STA 0,CODE       ; FOR OTHER ASSEMBLER ROUTINES
00062'001400          JMP 0,3          ; RETURN↓
                      ;
                      ;
00063'054417 SHL:     STA 3,SAV33      ; SAVE RETURN ADDRESS
00064'034417          LDA 3,ARRAY      ; POINTER
00065'175400          INC 3,3          ; +1
00066'030417          LDA 2,NWDS       ; NO. OF WORDS
00067'050415          STA 2,TEMP       ; TEMP STORAGE
00070'021400 NEXT:    LDA 0,0,3        ; GET WORD
00071'101122          MOVZL 0,0,SZC    ; SHIFT LEFT
00072'011777          ISZ -1,3         ; INC PREVIOUS WORD
00073'041400          STA 0,0,3        ; REPLACE WORD
00074'175400          INC 3,3          ; POINTER+1
00075'014407          DSZ TEMP         ; DONE YET
00076'000772          JMP NEXT
00077'034403          LDA 3,SAV33      ; RETURN
00100'001400          JMP 0,3          ; ZAP
                      ;
                      ; DATA
                      ;
00101'000000 SAV3:    0
00102'000000 SAV33:   0
00103'000000 ARRAY:   0
00104'000000 TEMP:    0
00105'000000 NWDS:    0
00106'000000 NBITS:   0
00107'077777 .DATL:   DATL
00110'000003 THREE:   3
00111'000000 CODE:    0
00112'010041 POLY:    10041
00113'077777 .DARAY:  DARAY
00114'077777 .WORK:   WORK
                      .END
  0003  FCRC
ARRAY   000103'
CODE    000111'
COPY    000007'
CRC     000000'
DARAY   000113'X
DATL    000107'X
DVD     000042'
FINIS   000056'
LP1     000031'
NBITS   000106'
NEXT    000070'
NWDS    000105'
POLY    000112'
SAV3    000101'
SAV33   000102'
SHL     000063'
TEMP    000104'
THREE   000110'
WORK    000114'X
.CPYL   000001$X
.DARA   000113'
.DATL   000107'
.FRET   000002$X
.WORK   000114'
End of Listing
```

Appendix 4

Micronova System Software

```
                        ;**********************************************
                        ;* READ A BLOK OF DATA FROM SLOW A-D'S        *
                        ;**********************************************
                XDBLOK: .TXT "(0)(5)DBLO"
04572'000005
04573'042102
04574'046117
04575'000000
04576'004350'BDBLOK:    MRUN
04577'075401 DBLOK:     PSHA 3
04600'071401            PSHA 2
04601'034065-           LDA 3,DVALS
04602'175400            INC 3,3
04603'024126-           LDA 1,NVALS
04604'124400            NEG 1,1
04605'102400            SUB 0,0
04606'062140 CVALS:     DOBS 0,40           ; SEND ADDX TO I/FACE
04607'065401            PSHA 1              ;SAVE ALL
04610'061401            PSHA 0
04611'075401            PSHA 3
04612'006430            JSR @RWAIT          ; 8 MILLISEC. DELAY
04613'000001            1                   ; THE DELAY
04614'075601            POPA 3              ; RESTORE ALL
04615'061601            POPA 0
04616'065601            POPA 1
04617'063540            SKPBZ 40            ; WAIT TILL DONE
04620'000412            JMP DB1             ; NOT DONE
04621'071440            DIB 2,40            ; GET CONVERTED VALUE
04622'051400            STA 2,0,3           ; STORE IT
04623'175400            INC 3,3             ; BUMP ARRAY POINTER
04624'101404            INC 0,0             ; BUMP ADDRESS
04625'125404            INC 1,1,SZR         ; BUMP COUNTER
04626'000752            JMP CVALS           ; REPEAT
04627'071601            POPA 2
04630'075601            POPA 3
04631'001400            JMP 0,3             ; FINISH
04632'075401 DB1:       PSHA 3              ;
04633'061401            PSHA 0              ; SAVE 3 AND 0
04634'020410            LDA 0,POUND
04635'024406            JSR @XOUT
04636'063240            NIOC 40
04637'061501            POPA 0              ; RESTORE 3 AND 0
04640'075601            POPA 3
04641'000745            JMP CVALS           ; TRY AGAIN
04642'006225'%WAIT:     WAIT
04643'001421'%OUT:      COUT
04644'000344 POUND:     44
+ 0091  SAVRE
```

```
                        ;*************************************
                        ;* COMPARE ARRAY VALUES WITH THE PRESET  *
                        ;* DATA BASE FORMAT IS :-                *
                        ;* HI,LO,ACTION IF HI, ACTION IF LO      *
                        ;*************************************
                XLOOKP: .TXT "(Ø)(5)LOOK"
 04645' 200025
 04646' 206121
 04647' 047515
 04650' 020020
 04651' 030572' 2DCO%P: XDBLCK
 04652' 071402  DCOMP:  PUSH 3
 04653' 071401          PUSH 2              ; SAVE ACC'S
 04654' 020127-MCSTART:         LDA 0,DBASE     ; ADDRESS OF DATA BASE
 04655' 040020          STA 0,@1PUT         ; AUTO INC 1
 04656' 020126-         LDA 0,NVALS         ; NO. OF VALUES
 04657' 040125-         STA 2,CUNT          ; COUNTER
 04660' 020035-         LDA 0,DVALC         ; DATA ARRAY POINTER
 04661' 040022          STA 0,@2PUT         ; STORE IN AUTO INCR
 04662' 020121  MCLUP:  LDA 0,@2PUT         ; GET FIRST VALUE
 04663' 101120          MOVS 0,0            ; SWAP BYTES
 04664' 024441          LDA 1,D177          ; MASK 177
 04665' 107420          ANDZ 0,1            ; ISOLATE BOTTOM 8 BITS
 04666' 036022          LDA 3,@D1AUT        ; GET DATA BASE VALUE
 04667' 136532          SUBZL# 1,3,SZC      ; MEASURED)MAX ?
 04670' 000432          JMP RANGE           ; YES SO XFER
 04671' 036022          LDA 3,@D1AUT        ; GET DATA BASE VALUE
 04672' 166532          SUBZL# 3,1,SZC      ; MEASURED(MIN ?
 04673' 000427          JMP RANGE           ; YES SO XFER
 04674' 101300          MOVS 0,0            ; SWAP BYTES BACK
 04675' 024430          LDA 1,D177          ; GET MASK
 04676' 107420          ANDZ 0,1            ; ISOLATE 8 BITS
 04677' 020427          LDA 0,D2            ; CONSTANT=2
 04700' 034022          LDA 3,D1AUT         ; DATA BASE POINTER
 04701' 117020          ADDZ 0,3            ; OFFSET TO NEXT VALUE
 04702' 054022          STA 3,D1AUT         ; STORE IT
 04703' 036022          LDA 3,@D1AUT        ; GET DATA BASE HI VALUE
 04704' 136532          SUBZL# 1,3,SZC      ; JMP IF ( MAX
 04705' 000415          JMP RANGE           ; OUTSIDE RANGE
 04706' 036022          LDA 3,@D1AUT        ; GET MIN VALUE FROM DATA BASE
 04707' 166532          SUBZL# 3,1,SZC      ; OUTSIDE RANGE ?
 04710' 000412          JMP RANGE           ; YES SO TAKE ACTION
 04711' 034022          LDA 3,D1AUT         ; GET DATA BASE VALUE
 04712' 020414          LDA 0,D2            ; OFFSET
 04713' 117020          ADDZ 0,3            ; POINTER TO NEXT VALUE
 04714' 054022          STA 3,D1AUT         ; STORE IT
 04715' 014166-         DSZ CUNT            ; DECREMENT COUNTER
 04716' 000744          JMP MCLUP           ; NEXT
 04717' 071601          POPA 2
 04720' 075601          POPA 3              ; LEAVE
 04721' 001400          JMP 0,3             ; BYE
                        ;******
 04722' 034022 RANGE:   LDA 3,D1AUT         ; DATA BASE VALUE
 04723' 007402          JSR @2,3            ; Z.A..A...A....A..... ER I MEAN CRASH
 04724' 000773          JMP .-5             ;GO HOME NOW

                        ;*************************************
                        ;* CONSTANTS NEEDED BY THIS ROUTINE  *
  0092  SABRE
                        ;*************************************
 04725' 000377 D177:    377
 04726' 000002 D2:      2
→0093  SABRE
```

```
                                ;**********************************************
                                ;* COMPUTER-COMPUTER XFER PROGRAM             *
                                ;**********************************************
                    XXFER:      .TXT "(0)(4)XFER"
 04727' 000004
 04730' 054106
 04731' 042522
 04732' 000000
 04733' 004645' BXFER:  XDCOMP
 04734' 020171-XFER:    LDA 0,ADX3          ; MICRO NOVA ADDRESS
 04735' 061260          DOAC 0,60           ; SET IT
 04736' 020172-         LDA 0,MADX          ; NOVA 3 ADDRESS
 04737' 062060          DOB 0,60            ; SET IT
 04740' 020173-         LDA 0,NWDS          ; NO OF WORDS
 04741' 063160          DOCS 0,60           ; SET AND RUN
 04742' 063560          SKPBZ 60            ; DONE ?
 04743" 000777          JMP .-1             ;NO
 04744' 001400          JMP 0,3             ; LEAVE
 04745' 177776 XCOU:    -2
+ 0094  SABRE
```

```
                        ;*****************************************
                        ;* POWER FAIL AUTO RESTART (PFART)       *
                        ;*****************************************
                XPFAIL: .TXT "(Ø)(5)PFAI"
04746'000005
04747'050106
04750'040511
04751'000000
04752'004727'BPFAIL:  XXFER
04753'075401 PFAIL:   PSHA 3
04754'071401          PSHA 2
04755'020177-         LDA Ø,KAPOW       ;ADDRESS OF NOVA POWERFAIL
04756'024525          LDA 1,UURUN       ;DO DA
04757'123020          ADDZ 1,Ø          ;GENAREATE JMP @KAPOW
04760'040532          STA Ø,NGO         ;STORE IT
04761'020520          LDA Ø,MIN7        ;MINUS 7
04762'040520          STA Ø,PFLAG       ;NUMBER OF ATTEMPTS TO START
04763'020524          LDA Ø,P1WAIT      ;WAIT LOOP 1
04764'040166-         STA Ø,CUNT        ;COUNTER 1
04765'020523          LDA Ø,P2WAIT      ;WAIT LOOP 2
04766'040167-         STA Ø,CUNT1       ;COUNTER 2
04767'020174-         LDA Ø,PNAVE       ;AVERAGES
04770'040172-         STA Ø,MADX        ;MICRO ADD
04771'040171-         STA Ø,ADX3        ;AGAIN
04772'102000          ADC Ø,Ø           ;-1
04773'040173-         STA Ø,NWDS        ; NUMBER OF WORDS
04774'006504 NWVAL:   JSR @.XFER        ;GET IT
04775'024032-         LDA 1,NAVE        ;NUMBER OF AVERAGES
04776'006502          JSR @.XFER        ;WORD AGAIN
04777'020032-         LDA Ø,NAVE        ;GET IT
05000'106435          SUBZ# Ø,1,SNR     ; SAME
05001'000402          JMP TRYAG         ;AGAIN
05002'000446          JMP TXS1          ; GO MAM GO
05003'014166-TRYAG:   DSZ CUNT          ;DEC COUNET 1
05004'000770          JMP NWVAL         ;AND AGAIN
05005'020502          LDA Ø,P1WAIT      ;WAIT ONE
05006'040166-         STA Ø,CUNT        ;STORE
05007'014167-         DSZ CUNT1         ;TRIED ENOUGH ?
05010'000764          JMP NWVAL         ;NO
                      ; NO CHANGE IN AVERAGE SO LOOK AT LOC NOVA
05011'020503 PF1:     LDA Ø,NENQ        ;ENQUIRE STATUS
05012'040145-         STA Ø,MICRO       ; STORE IT
05013'006500          JSR @MIC9         ;SEND IT
05014'006470          JSR @WAIA         ;WAIT FOR 20 SECONDS
05015'020215          20215
05016'004545          JSR NOVDM         ;GET LOC NOVA FROM NOVA
05017'000434          JMP MEMDEM        ;NOT SET
05020'101222          MOVZR Ø,Ø,SZC     ;IS BIT 15 SET ?
05021'000411          JMP WAITB         ;YES
05022'101222          MOVZR Ø,Ø,SZC     ;ALREADY THERE ?
05023'000424          JMP TXST          ;YES SO GO LOOKING
05024'101222          MOVZR Ø,Ø,SZC     ;EOT ?
05025'006461          JSR @AEMES        ;YES TELL LEICESTER
05026'101222          MOVZR Ø,Ø,SZC     ;RUNNING ?
05027'000421          JMP TXS1          ;YES
05030'101222          MOVZR Ø,Ø,SZC     ; WAITIN ?
05031'002176-         JMP @.MON         ;YES RETURN TO MONITOR
05032'006452 WAITB:   JSR @WAIA         ;WAIT Ø.5 SECONDS
0095  SABRE
```

```
05033'000246            246
05034'004530            JSR NOVDM       ;GET IT
05035'000775            JMP WAITB       ;NOT READY SO WAIT
05036'101220            MOVZR 0,0
05037'101222            MOVZR 0,0,SZC   ;READY
05040'000407            JMP TXST        ;YES
05041'101222            MOVZR 0,0,SZC   ; EOT ?
05042'006444            JSR @AEMES      ;YES
05043'101222            MOVZR 0,0,SZC   ; RUNNING ?
05044'000404            JMP TXS1        ;YES
05045'101222            MOVZR 0,0,SZC   ;WAITING
05046'002176-           JMP @.MON       ;YES RETURN TO MONITOR
05047'006436 TXST:      JSR @ATXLK      ;LOOK
05050'071601 TXS1:      POPA 2
05051'075601            POPA 3
05052'001400            JMP 0,3         ;BYE BYE BABY
                        ;
                        ;THIS BIT DMAS MEMORY TO NOVA
                        ;
05053'010427 MEMDEM:    ISZ PFLAG       ;TRIED 7 TIMES ?
05054'000402            JMP MEM1                ;TRY AGAIN
05055'000440            JMP SFAIL       ;FAILED TEKK THE WORLD
05056'102400 MEM1:      SUB 0,0         ;0
05057'101240            MOVOR 0,0       ;100000
05060'101400            INC 0,0         ;100001
05061'040171-           STA 0,ADX3      ;NOVA ADDRESS
05062'101400            INC 0,0         ;NUMBER OF WORDS
05063'040173-           STA 0,NWDS      ;STORE
05064'102520            SUBZL 0,0       ; 1
05065'101120            MOVZL 0,0       ;2
05066'040172-           STA 0,MADX      ;MICRONOVA ADDRESS
05067'006411            JSR @.XFER      ;DO THE XFER
05070'102000            ADC 0,0         ;-1
05071'040171-           STA 0,ADX3      ;NOVA ADDRESS
05072'020417            LDA 0,CGO       ;LOC OF TRANSFER
05073'040172-           STA 0,MADX      ;MICRO ADDRESS
05074'102000            ADC 0,0         ;-1
05075'040173-           STA 0,NWDS      ;STORE
05076'006402            JSR @.XFER      ;DO IT
05077'000712            JMP PF1         ;BACK AGAIN
                        ;*****************************************
                        ;* BITS AND PIECES NEEDED HERE           *
                        ;*****************************************
05100'004734'.XFER:     XFER
05101'177771 MIN7:      -7
05102'000000 PFLAG:     0
05103'002000 UURUN:     2000
05104'002205'WAIA:      WAIT
05105'005774'ATXLK:     TXLOOK
05106'006165'AEMES:     EOTMES
05107'000000 P1WAIT:    0
05110'000007 P2WAIT:    7
05111'005112'CGO:       .+1
05112'000000 NGO:       0
05113'005572'MIC9:      MICDMA
05114'100100 NENG:      100100
→ 0096  SABRE
```

```
                        ;**************************************
                        ; JUMP HERE WHEN CANNOT START
                        ;**************************************
05115'020406 SFAIL:     LDA 0,FMES          ;ADD OF MESSAGE
05116'024440            LDA 1,FLEN          ;LENGTH
05117'006403            JSR @BTY4           ;PRINT IT
05120'002176-           JMP @.MON           ;RETURN TO MONITOR
05121'000262'BPU4:      PUSH
05122'006146'BTY4:      CTYPE
05123'012250"FMES:      .+1*2
                        .TXT "FAILED ON POWER RESTART PLEASE INFORM LEICESTER <1
05124'043101
05125'044514
05126'042504
05127'020117
05130'047040
05131'050117
05132'053505
05133'051040
05134'051105
05135'051524
05136'040522
05137'052040
05140'050114
05141'042501
05142'051505
05143'020111
05144'047106
05145'047522
05146'046440
05147'046105
05150'044503
05151'042523
05152'052105
05153'051040
05154'006412
05155'000000
05156'000062 FLEN:      50.

→ 0097  SABRE
```

```
                                ;**********************************
                                ;THIS ROUTINE READS NOVA FROM NOVA
                                ;**********************************
                    XNOVD:      .TXT "(0)(5)NOVD"
    05157'000005
    05160'047117
    05161'053104
    05162'000000
    05163'004746'BNOVD:         XPFAIL
    05164'075401 NOVDM:         PSHA 3
    05165'071401                PSHA 2              ;SAVE ACCS
    05166'020432                LDA 0,BNOVA         ;ADDRESS OF NOVA
    05167'040172-               STA 0,MADX          ;UNOVA ADD
    05170'040171-               STA 0,ADX3          ;NOVA 3 ADD
    05171'102000                ADC 0,0             ;-1
    05172'040173-               STA 0,NWDS          ;1 WORD
    05173'006705                JSR @.XFER          ;DO IT
    05174'020146-               LDA 0,NOVA          ;GET IT
    05175'101113                MOVL# 0,0,SNC       ;SET ?
    05176'000417                JMP XNV1            ;HOME
    05177'061401                PSHA 0              ;SAVE IT
    05200'102400                SUB 0,0             ;0
    05201'040146-               STA 0,NOVA          ; STORE IT
    05202'020416                LDA 0,BNOVA         ;ADDRESS
    05203'100400                NEG 0,0
    05204'102000                COM 0,0             ;SUB -1
    05205'024414                LDA 1,TNTT          ;100000
    05206'123020                ADDZ 1,0            ;ADD
    05207'040171-               STA 0,ADX3          ;MICRO ADD
    05210'006670                JSR @.XFER          ; DO IT
    05211'061601                POPA 0   ;RESTORE 0
    05212'071601                POPA 2              ;RESTORE ACCS
    05213'075601                POPA 3
    05214'001401                JMP 1,3             ;HOME
    05215'071601 XNV1:          POPA 2
    05216'075601                POPA 3
    05217'001400                JMP 0,3             ;HOME
    05220'000146-BNOVA:         NOVA
    05221'100000 TNTT:          100000
→ 0098  SABRE

                                ;***********************************************
                                ;** ACTION TO BE TAKEN ON ERROR VOLTAGES     **
                                ;***********************************************
                    XACT1:      .TXT "(0)(4)ACT1"
    05222'000004
    05223'040503
    05224'052061
    05225'000000
    05226'005157'BACT1:         XNOVD
    05227'075401 ACT1:          PSHA 3
    05230'071401                PSHA 2
    05231'020422                LDA 0,052           ; STAR
    05232'006420                JSR @.CO20          ; COUT
    05233'071601                POPA 2
    05234'075601                POPA 3
    05235'001400                JMP 0,3
                    XACT2:      .TXT "(0)(4)ACT2"
    05236'000004
    05237'040503
    05240'052062
    05241'000000
    05242'005222'BACT2:         XACT1
    05243'075401 ACT2:          PSHA 3
    05244'071401                PSHA 2
    05245'020407                LDA 0,053           ; +
    05246'006404                JSR @.CO20          ; COUT
    05247'071601                POPA 2
    05250'075601                POPA 3
    05251'001400                JMP 0,3
```

```
                        ;*************************************************
                        ;** WHAT WE NEED HERE IS NOT MUCH !!!!!!!!     **
                        ;*************************************************
 05252'001421'.C020:    COUT
 05253'000052 052:      52
 05254'000100 053:      100
→ 0099  SABRE

                        ;*************************************************
                        ;* BUBBLE MEMORY PROGRAMS                        *
                        ;* N.B.  NIOS 20 TAKES THE A0 LINE HI            *
                        ;*       NIOC 20 TAKES THE A0 LINE LO            *
                        ;*       DOA WILL SEND A WRITE PULSE             *
                        ;*       DIA WILL LATCH THE READ DATA            *
                        ;*       DIC WILL READ THE DATA                  *
                        ;*************************************************
               XRBUB:   .TXT "<0><5>RPAG"
 05255'000005
 05256'051120
 05257'040507
 05260'000000
 05261'005236'BRBUB:    XACT2
 05262'075401 RBUB:     PSHA 3
 05263'071401           PSHA 2
 05264'034464           LDA 3,N1BYTE      ; NO OF BYTES TO READ
 05265'174400           NEG 3,3           ; 2'S
 05266'030463           LDA 2,BUBUF       ; BUBBLE BUFFER
 05267'050022           STA 2,D1AUT       ; STORE AT AUTO INC LOCN
 05270'020456           LDA 0,BUBR        ; BUBBLE READ COMMAND
 05271'060120           NIOS 20           ; A0=HI
 05272'061020           DOA 0,20          ; WRITE DATA A0=LO
 05273'060220           NIOC 20
 05274'060120 NXBTE:    NIOS 20           ; A0=HI
 05275'060420           DIA 0,20          ; READ DATA TO LATCH A0=LO
 05276'060220           NIOC 20
 05277'062420           DIC 0,20          ; GET DATA
 05300'101223           MOVZR 0,0,SNC     ; SKIP IF FIFO HAS DATA
 05301'000773           JMP NXBTE         ; READ STATUS AGAIN
 05302'060420           DIA 0,20          ; READ DATA
 05303'062420           DIC 0,20          ; INTO MACHINE
 05304'042022           STA 0,@D1AUT      ; AUTO INC
 05305'175404           INC 3,3,SZR       ; FINISHED ?
 05306'000766           JMP NXBTE         ; GET NEXT
 05307'071601           POPA 2            ; OK
 05310'075601           POPA 3            ; DONE
 05311'001400           JMP 0,3
→ 0100  SABRE
```

```
                        ;*************************************************
                        ;* THE MAIN MICRONOVA RUN ROUTINE                *
                        ;*************************************************
              XMMAIN:   .TXT "(Ø)(4)MAIN"
 05453'000004
 05454'046501
 05455'044516
 05456'000000
 05457'005312'BMMAIN:   XWBUB
 05460'060277 MMAIN:    INTDS
 05461'101000           MOV Ø,Ø          ; WAIT TILL ION=OFF
 05462'020204-          LDA Ø,MSKØ       ; MASK OUT
 05463'062077           DOB Ø,CPU        ; ALL OFF
 05464'101000           MOV Ø,Ø          ; WAIT TILL MSK OUT
 05465'020205-          LDA Ø,INT1       ; GET ADDRESS
 05466'176520           SUBZL 3,3        ; GENERATE +1
 05467'041400           STA Ø,Ø,3        ; STORE IT
 05470'060134 INIT1:    NIOS ULM         ; INITIALISE ULM
 05471'020201-          LDA Ø,SL         ; SYNC LINE
 05472'024202-          LDA 1,LC         ; CHARACTERISTICS
 05473'030210-          LDA 2,SYN        ; SYNC CHAR
 05474'063534           SKPBZ ULM        ; INIT DONE ?
 05475'000777           JMP .-1 ; WELL ?
 05476'061034           DOA Ø,ULM        ; SET LINE
 05477'067034           DOC 1,ULM        ; SET CHARACTERISTICS
 05500'073034           DOC 2,ULM        ; SET CHARS
 05501'101400           INC Ø,Ø          ; SET TX SECTION
 05502'061034           DOA Ø,ULM        ; DONE
 05503'073034           DOC 2,ULM        ; SET SYNC
 05504'024457           LDA 1,MOFF              ;MODEM OFF
 05505'067034           DOC 1,ULM               ;SEND IT
                        ;INTEN
 05506'102400           SUB Ø,Ø          ; +Ø
 05507'040206-          STA Ø,EOM        ; END OF MESSAGE
 05510'040207-          STA Ø,LEM        ; END OF MESSAGE
 05511'101400           INC Ø,Ø          ; +1
 05512'040216-          STA Ø,SINK       ; SET SINK=ON
 05513'020011-          LDA Ø,STORE      ; GET POINTER TO BUFFER
 05514'040001-          STA Ø,WE         ; FOR WORD ROUTINE
 05515'102400           SUB Ø,Ø          ; +Ø
 05516'040005-          STA Ø,ERR        ; ERRORS=OFF
 05517'006442           JSR @.PFAIL      ; POWER FAIL
 05520'006437 LUPN:     JSR @.DBLOK      ; GET BLOK OF DATA FROM SLOW A-D'S
 05521'006437           JSR @.DCOMP      ; COMPARE
 05522'004572           JSR DKDMA        ;TIME TO DMA ?
 05523'101000           MOV Ø,Ø          ;DUMMY
 05524'006726           JSR @ANVDM       ;YES READ NOVA STATUS
 05525'000414           JMP LP1 ;NOTHING TO REPORT
 05526'101222           MOVZR Ø,Ø,SZC    ;IS IT STARTING ?
 05527'000417           JMP LP3          ; YES
 05530'101222           MOVZR Ø,Ø,SZC    ;ABOUT TO SART SEQUENCE ?
 05531'000422           JMP LP4          ; YES
 05532'101222           MOVZR Ø,Ø,SZC    ; EOT ??
 05533'006427           JSR @AEOM        ;YES TELL LEICSTER
 05534'101222           MOVZR Ø,Ø,SZC    ;RUNNING ?
 05535'000763           JMP LUPN         ;WAST OF TIME
 05536'101222           MOVZR Ø,Ø,SZC    ;WAITING ?
 05537'002176-          JMP @.MON        ; ERR SHOULD NOT BE HERE
0102  SABRE
 05540'000760           JMP LUPN         ;NEITHER BACK AGAIN
 05541'053610 LP1:      SKPDN TTI        ; CHAR FROM TERMINAL ?
 05542'000756           JMP LUPN         ;NO BACK AGAIN
 05543'060610           DIAC Ø,TTI       ;GET CHAR
 05544'005420           JSR @AWREAD      ;PRINT READY
 05545'002176-          JMP @.MON        ;YES MONITOR
 05546'034410 LP3:      LDA 3,LP33       ;RETURN ADDRESS FROM PFAIL
 05547'075401           PSHA 3           ;ON STACK
 05550'071401           PSHA 2           ;PUSHH ACC 2
 05551'002401           JMP @.+1         ;GO TO PF1
 05552'305032'          WAITB            ;WAIT FOR EOT FIND
 05553'006402 LP4:      JSR @L44         ;TXLOOK ROUTINE
 05554'020744           JMP LUPN         ;BACK AGAIN
 05555'305774'L44:      TXLOOK
 05556'005520'LP33:     LUPN
```

```
                        ;*****************************************
                        ;* BUBBLE WRITE PAGE COMMAND             *
                        ;*****************************************
                XWBUB:  .TXT "(0)(5)WPAG"
    05312'000005
    05313'053520
    05314'040507
    05315'000000
    05316'005255'BWBUB: XRBUB
    05317'075401 WBUB:  PSHA 3
    05320'071401        PSHA 2
    05321'034427        LDA 3,N1BYTE    ; NO OF BYTES=64=1 PAGE
    05322'174400        NEG 3,3 ; 2'S
    05323'030426        LDA 2,BUBUF     ; BUFFER
    05324'050022        STA 2,D1AUT     ; AUTO INC 1
    05325'020422        LDA 0,BUBW      ; BUBBLE WRITE COMMAND
    05326'060120        NIOS 20         ; A0=HI
    05327'061020        DOA 0,20        ; WRITE DATA A0=LO
    05330'060220        NIOC 20
    05331'060120 NYBTE: NIOS 20         ; A0=HI
    05332'060420        DIA 0,20        ; GET STATUS
    05333'060220        NIOC 20
    05334'062420        DIC 0,20        ; INTO MACHINE
    05335'101223        MOVZR 0,0,SNC   ; FIFO READY ?
    05336'000773        JMP NYBTE       ; NO SO WAIT
    05337'022022        LDA 0,@D1AUT    ; GET A DATUM
    05340'061020        DOA 0,20        ; WRITE IT
    05341'175404        INC 3,3,SZR     ; FINISHED ?
    05342'000767        JMP NYBTE       ; NO
    05343'071601        POPA 2          ; FINISH
    05344'075601        POPA 3
    05345'001400        JMP 0,3

                        ;*************************************
                        ;* BUBBLE COMMANDS ETC.              *
                        ;*************************************
    05346'000022 BUBR:  22
    05347'000023 BUBW:  23
    05350'000100 N1BYTE: 100
    05351'005351'BUBUF: .
          000100        .BLK 64.
    05452'005164'ANVDM: NOVDM
→ 0101   SABRE
```

```
                        ;*****************************************
                        ;* THINGS NEEDED FOR THIS ONE O.K.       *
                        ;*****************************************
  05557'004577'.DBLOK:  DBLOK
  05560'004652'.DCOMP:  DCOMP
  05561'004753'.PFAIL:  PFAIL
  05562'006165'AEOM:    EOTMES
  05563'020001 MOFF:    20001    ;MODEM OFF
  05564'006205'AWREAD:  WREAD
→ 0103  SABRE

                        ;*************************************
                        ;THIS ROUTINE DMAS LOC MICRO TO NOVA
                        ;*************************************
              XMICD:    .TXT "(0)(6)MICD"
  05565'000006
  05566'046511
  05567'041504
  05570'000000
  05571'005453'BMICD:   XMMAIN
  05572'075401 MICDMA:  PSHA 3
  05573'071401          PSHA 2           ;SAVE ACCS
  05574'020416          LDA 0,BRO        ;MICRO ADD
  05575'040172-         STA 0,MADX       ;STORE IT
  05576'100400          NEG 0,0
  05577'100000          COM 0,0          ;SUB 1
  05600'024411          LDA 1,TENTT      ;100000
  05601'123020          ADDZ 1,0         ;ADD
  05602'040171-         STA 0,ADX3       ;NOVA ADDRESS
  05603'102000          ADC 0,0          ;-1
  05604'040173-         STA 0,NWDS       ;1 WOR D TRANSDER
  05605'006506          JSR @ZXFER       ;DO IT
  05606'071601          POPA 2
  05607'075601          POPA 3           ;RESTORE ACCS
  05610'001400          JMP 0,3          ;HOME
  05611'100000 TENTT:   100000
  05612'000145-BRO:     MICRO
→ 0104  SABRE
```

```
                        ;*****************************************
                        ;THIS ROUTINE DMAS THE CURRENT TIME AND
                        ;PUTS IT INTO MICRO NOVA
                        ;*****************************************
                XTIMD:  .TXT "(0)(6)TIMD"
05613'000006
05614'052111
05615'046504
05616'000000
05617'005565'BTIMD:     XMICD
05620'075401 TIMDMA:    PSHA 3
05621'071401            PSHA 2          ;SAVE ACCS
05622'020470            LDA 0,TTF       ;NUMBER OF WORDS TO DMA
05623'100400            NEG 0,0         ; 2S COMP
05624'040173-           STA 0,NWDS      ; STORE IT
05625'020014-           LDA 0,TBUF      ;ADDRESS BUFFER
05626'101400            INC 0,0         ;+1

05627'040172-           STA 0,MADX      ;MICRO ADD
05630'040171-           STA 0,ADX3      ;NOVA ADDRESS
05631'006462            JSR @ZXFER      ; DMA
                        ;
                        ; CONVERT TO RELA TIME
                        ;
05632'034457            LDA 3,PSTR1     ;POINTER TO PUT SECONDS
05633'030014-           LDA 2,TBUF      ;POINTER TO STORE
05634'020451            LDA 0,MIN4      ;MINUS 4 COUNTER D H M S
05635'040451            STA 0,DPONT     ;STORE IT
05636'025002 TIMLOP:    LDA 1,2,2       ;GET TENS
05637'121000            MOV 1,0
05640'127120            ADDZL 1,1
05641'107120            ADDZL 0,1       ;MULT BY 10
05642'021001            LDA 0,1,2       ;GET UNITS
05643'107020            ADDZ 0,1        ;GET TOTAL
05644'045400            STA 1,0,3       ;AND STORE
05645'174400            NEG 3,3
05646'174000            COM 3,3         ;SUB -1
05647'151400            INC 2,2
05650'151400            INC 2,2         ;INC POINTER
05651'010435            ISZ DPONT       ;DONE
05652'000764            JMP TIMLOP      ;NO
05653'151400            INC 2,2         ;RIGHT PLACE FOR MONTH
05654'025002            LDA 1,2,2       ;YES GET 10 MONTHS
05655'121000            MOV 1,0
05656'127120            ADDZL 1,1
05657'107120            ADDZL 0,1       ;* 10
05660'021001            LDA 0,1,2       ;UNITS
05661'107020            ADDZ 0,1        ;GET RESULT
05662'102520            SUBZL 0,0       ;GEN +1
05663'034424            LDA 3,MTAB      ;POINTER TO MONTH TABLE
05664'152400            SUB 2,2         ;ACC NUMBER OF DAYS = 0
05665'106435 DCAY:      SUBZ# 0,1,SNR   ;FINISHED LOOKING ?
05666'000410            JMP CDONE       ;YES
05667'061401            PSHA 0          ;AWAY
05670'021400            LDA 0,0,3       ;NUMBER OF DAYS IN MONTH
05671'113020            ADDZ 0,2        ;ADD IN
05672'175400            INC 3,3         ;INC POINTER
05673'061601            POPA 0  ;RESTORE POINTER
0105  SABRE
```

```
05674'101400            INC 0,0         ;NEXT MONTH
05675'000770            JMP DCAY        ;BACK AGIAN
05676'034412 CDONE:     LDA 3,PSTOR     ;POINTER
05677'021400            LDA 0,0,3       ;NUMBER OF DAYS IN CURRENT MONTH
05700'143020            ADDZ 2,0        ;ADD
05701'041400            STA 0,0,3       ;STORE IT
05702'071601            POPA 2
05703'075601            POPA 3          ;RESTORE
05704'001400            JMP 0,3         ;HOME
05705'177774 MIN4:      -4
05706'000000 DPONT:     0
05707'002573'MTAB:      MTHTB+1
05710'000101-PSTOR:     DVALS+12.
05711'000104-PSTR1:     DVALS+15.
05712'000014 TTF:       14
05713'004734'ZXFER:     XFER
→ 0106  SABRE

                        ;*************************************************
                        ;THIS ROUTINE CHECKS TO SEE IF TIME TO DMA
                        ;*************************************************
05714'054427 DKDMA:     STA 3,OKRET     ;USUAL
05715'004703            JSR TIMDMA      ;GET TIME
05716'034772            LDA 3,PSTOR     ;ADDRESS OF TIME
05717'020031-           LDA 0,INTTIM    ;INT TIME
05720'101220            MOVZR 0,0       ;DIVIDE BY 2
05721'105000            MOV 0,1         ; * 10
05722'103120            ADDZL 0,0
05723'123120            ADDZL 1,0
05724'031403 DK1:       LDA 2,3,3       ;GET SECONDS
05725'142415            SUB# 2,0,SNR    ;EQUAL ?
05726'000412            JMP NOWF        ;YES
05727'024031-           LDA 1,INTTIM    ;INT TIME
05730'131000            MOV 1,2
05731'127120            ADDZL 1,1
05732'147120            ADDZL 2,1       ; * 10
05733'123020            ADDZ 1,0        ;ADD
05734'024410            LDA 1,SIXTY     ;60
05735'106032            ADCZ# 0,1,SZC   ;SKIP IF )= 60
05736'000766            JMP DK1         ;TRY AGAIN
05737'002404            JMP @OKRET      ;NOT THIS TIME BYE
05740'004412 NOWF:      JSR DMVALS      ;DO IT
05741'034402            LDA 3,OKRET     ;RETURN ADDRESS +1
05742'001401            JMP 1,3         ;HOME
05743'000000 OKRET:     0
05744'000074 SIXTY:     60.

→ 0107  SABRE
```

```
, → 0109  SABRE

                        ;*************************************
                        ;THIS NON SUBROUTINE LOOKS AT TX COMING ON
                        ;*************************************************
05774'054511 TXLOOK:    STA 3,AAAA          ;RETURN ADD
05775'006506            JSR @MWAIT          ;5 SECONDS WAIT
05776'003750            3750
05777'006507            JSR @XBLOK          ;READ SLOW AD
06000'030065-           LDA 2,DVALS         ;POINTER
06001'036503            LDA 3,@ADBST        ;MAX MIN POINTERS
06002'021001            LDA 0,1,2           ; 24VOLTS STATUS
06003'025404            LDA 1,4,3           ;MAX VALUE
06004'031405            LDA 2,5,3           ;MIN VALUE
06005'004502            JSR COMP1           ;COMPARE
06006'021004            LDA 0,4,2           ;DRIVER AIR FLOW
06007'025434            LDA 1,34,3          ;MAX VALUE
06010'031435            LDA 2,35,3          ;MIN VALUE
06011'004476            JSR COMP1           ;COMPARE
06012'021005            LDA 0,5,2           ;13 VOLTS MON
06013'025444            LDA 1,44,3          ;MAX
06014'031445            LDA 2,45,3          ;MIN
06015'004472            JSR COMP1           ;COMPARE
06016'021006            LDA 0,6,2           ;SENSE INHIBIT
06017'101300            MOVS 0,0            ;SWAP
06020'025450            LDA 1,50,3          ;MAX
06021'031451            LDA 2,51,3          ;MIN
06022'004465            JSR COMP1           ;COMPARE
06023'021010            LDA 0,10,2          ;PA AIR FLOW
06024'101300            MOVS 0,0            ;SWAP
06025'025470            LDA 1,70,3          ;MAX
06026'031471            LDA 2,71,3          ;MIN
06027'004460            JSR COMP1           ;COMPARE
06030'021010            LDA 0,10,2          ;AUTO ?
06031'025474            LDA 1,74,3          ;MAX
06032'031475            LDA 2,75,3          ;MIN
06033'004454            JSR COMP1           ;COMPARE
06034'021011            LDA 0,11,2          ;RX POWER SUPPLY
06035'101300            MOVS 0,0
06036'025500            LDA 1,100,3         ;MAX
06037'031501            LDA 2,101,3         ;MIN
06040'004447            JSR COMP1           ;COMPARE
06041'006442            JSR @MWAIT
06042'114061            114061
06043'006443            JSR @XBLOK          ;GET NEW BLOCK
                        ;
                        ; NOW CHECK DRIVER EHT HEATER DITTO PA
                        ;

06044'030065-           LDA 2,DVALS         ;POINTERRS
06045'036437            LDA 3,@ADBST
06046'021003            LDA 0,3,2           ;DRIVER EHT
06047'101300            MOVS 0,0            ;SWAP
06050'025420            LDA 1,20,3          ;MAX
06051'031421            LDA 2,21,3          ;MIN
06052'004435            JSR COMP1           ;COMPARE
06053'021004            LDA 0,4,2           ;DRIVER HEATERS
06054'101300            MOVS 0,0            ;SWAP
06055'025430            LDA 1,30,3          ;MAX
06056'031431            LDA 2,31,3          ;MIN
0110  SABRE
```

```
                        ;*************************************************
                        ;THIS ROUTINE DMAS VOLTAGES AND TIMES
                        ;STORED IN DVALS TO THE NOVA
                        ;*************************************************
                XDMVAL: .TXT "<0><6>DMVA"
  05745'000006
  05746'042115
  05747'053101
  05750'000000
  05751'005613' BDMV:   XTIMD
  05752'075401  DMVALS: PSHA 3
  05753'071401          PSHA 2          ;SAVE ACCS
  05754'020417          LDA 0,D40       ;NUMBER OF WORDS
  05755'040173-         STA 0,NWDS      ;STORE
  05756'020065-         LDA 0,DVALS     ;START ADDRESS
  05757'101400          INC 0,0
  05760'040172-         STA 0,MADX      ;MICRO ADDRESS
  05761'100400          NEG 0,0
  05762'100000          COM 0,0         ;SUB 1
  05763'024626          LDA 1,TENTT     ;100000
  05764'123020          ADDZ 1,0        ;ADD
  05765'040171-         STA 0,ADX3      ;STORE NOVA
  05766'006404          JSR @PXXFER     ;XFER
  05767'071601          POPA 2
  05770'075601          POPA 3
  05771'001400          JMP 0,3         ;HOME
  05772'004734' PXXFER: XFER
  05773'177740 D40:     -40
→ 0108  SABRE
```

```
                        ;*************************************************
                        ;
                        ; THIS ROOUTINE INFORMS LEICSTER WHAT IS HAP
                        ;
                        ;*************************************************
06145'001400 INFORM:    JMP 0,3 ;HOME
                        ;*************************************
                        ; THIS ROUTINE PRINTS OUT THE N CHA*
                        ; POINTED TO BY ACC 1 AND AT ADD
                        ; POINTED TO BY ACC 0              *
                        ;*************************************
06146'075401 CTYPE:     PSHA 3          ;SAVE RETURN ADDRESS
06147'071401            PSHA 2          ;AND ACC 2
06150'125005            MOV 1,1,SNR     ;ZERO LENGTH
06151'000407            JMP CNOT        ;YES
06152'040003-           STA 0,GSTART    ;STORE BYTE POINTER
06153'152520            SUBZL 2,2       ;GENERATE +1
06154'006407 CMORE:     JSR @ABY3       ;GET A BYTE
06155'006407            JSR @ACO3       ;PRINT IT
06156'146404            SUB 2,1,SZR     ;DONE
06157'000775            JMP CMORE       ;OBVIOUSLY NO
06160'071601 CNOT:      POPA 2          ;RESTORE ACC 2
06161'075601            POPA 3          ;ACC 3
06162'001400            JMP^0,3         ;HOME IS WHERE ACC 3 IS
06163'000051'ABY3:      BYTE
06164'001421'ACO3:      COUT
→ 0114  SABRE

                        ;*************************************************
                        ;
                        ; THIS HAPPENS WHEN EOT FOUND
                        ;
                        ;*************************************************
06165'054417 EOTMES:    STA 3,GGGG      ;RETURN ADDRESS
06166'004745            JSR TXFF        ;SWITCH OFF TX
06167'020405            LDA 0,MESEOT    ;ADD OF MESSAGE
06170'024403            LDA 1,MESLE     ; LENGTH
06171'004755            JSR CTYPE       ;PRINT IT
06172'002176-           JMP @.MON       ; MONITOR
06173'000015 MESLE:     15
06174'014372"MESEOT:    .+1*2
                        .TXT "(15)(12)EOT FOUND(15)(12)"
06175'006412
06176'042517
06177'052040
06200'043117
06201'052516
06202'042015
06203'005000
06204'000000 GGGG:      0
                        ;*************************************
                        ; TYPE READY                        *
                        ;*************************************
06205'054777 WREAD:     STA 3,GGGG      ;RETURN ADDRESS
06206'020405            LDA 0,REDMES    ;MESAAGE ADD
06207'024411            LDA 1,REDLEN    ;LENGTH
06210'004736            JSR CTYPE       ;PRINT IT
06211'034773            LDA 3,GGGG
06212'001400            JMP 0,3         ;HOME
06213'014430"REDMES:    .+1*2
                        .TXT "READY(15)(12)"
06214'051105
06215'040504
06216'054415
06217'005000
06220'000007 REDLEN:    7
→ 0115  SABRE
```

```
06057'004430            JSR COMP1       ;COMPARE
06060'021006            LDA 0,6,2       ;PA HEATER
06061'025454            LDA 1,54,3      ;MAX
06062'031455            LDA 2,55,3      ;MIN
06063'004424            JSR COMP1       ;COMPARE
06064'021007            LDA 0,7,2       ;PA HEATER
06065'025464            LDA 1,64,3      ;MAX
06066'031465            LDA 2,65,3      ;MIN
06067'004420            JSR COMP1       ;COMPARE
                        ;
                        ;FURTHER WIAT FOR POWER
                        ;
06070'006413            JSR @MWAIT
06071'076145            76145    ;90 SECONDS
06072'006414            JSR @XBLOK      ;NEXT BLOCK
06073'030065-           LDA 2,DVALS     ;POINTERS
06074'036410            LDA 3,@ADBST
06075'021005            LDA 0,5,2       ;POWER OUT
06076'101300            MOVS 0,0        ;SWAP
06077'025440            LDA 1,40,3      ;MAX
06100'031441            LDA 2,41,3      ;MIN
06101'004406            JSR COMP1       ;COMPARE
06102'002403            JMP @AAAA       ;BACK  HOME
06103'002206'MWAIT:     WAIT
06104'040765 ADBST:     DBST
06105'000000 AAAA:      0
06106'004577'XBLOK:     DBLOK
→ 0111  SABRE

                        ;*********************************
                        ; COMPARE ROUTINE
                        ;*********************************
06107'054415 COMP1:     STA 3,COMRET    ;HOME ADDRESS
06110'034415            LDA 3,T177      ;MASK
06111'163400            AND 3,0         ;GET BOTTOM 8 BITS
06112'106532            SUBZL# 0,1,SZC  ; ? ) MAX
06113'000406            JMP TXERR       ; YES
06114'142532            SUBZL# 2,0,SZC  ; ( MIN ?
06115'000404            JMP TXERR       ;YES
06116'036766 COMP2:     LDA 3,@ADBST    ;RESTORE POINTERS
06117'030065-           LDA 2,DVALS
06120'002404            JMP @COMRET     ;RETURN
06121'004412 TXERR:     JSR TXFF        ;OFF WITH TX
06122'004423            JSR INFORM
06123'000773            JMP COMP2
06124'000000 COMRET:    0
06125'000377 T177:      377
→ 0112  SABRE

                        ;*************************************************
                        ;
                        ; THIS ROUTINE SWITCHES TX OFF
                        ;
                        ;*************************************************
             XTXFF:     .TXT "(0)(4)TXFF"
06126'000004
06127'052130
06130'043106
06131'000000
06132'005745'BTXFF:     XDMVAL
06133'075401 TXFF:      PSHA 3
06134'071401            PSHA 2
06135'020407            LDA 0,TXF1      ;THING TO STORE
06136'040145-           STA 0,MICRO     ;STORE IT
06137'006404            JSR @AMIC       ; DO IT
06140'071601            POPA 2
06141'075601            POPA 3
06142'001400            JMP 0,3
06143'005572'AMIC:      MICDMA
06144'100040 TXF1:      100040
→ 0113  SABRE
```

```
      000040          .BLK 40
00126-000011 NVALS:   11
00127-040765 DBASE:   DBST      ; DATA BASE START
00130-000000 PMGAP:   0         ; NBEAM*PGAP=POINT OFFSET IN DOPPLER ARRAY
00131-045000 POWER:   45000     ; START OF RAW POWER DATA
00132-000000 DOPPLER: 0         ; START OF DOPPLER ARRAY
00133-000000 PARAY:   0         ; START OF SUMMED POWER ARRAY
00134-000000 DARAY:   0         ; START OF SUMMED DOPPLER ARRAY
00135-000000 PSAM:    0         ; NO. OF RAW POWER SAMPLES=NBEAM*NRANG
00136-000000 DSAM:    0         ; NO. OF RAW DOPPLER SAMPLES=PSAM+PMGAP
00137-000000 DSAM1:   0         ; NO. OF DOUBLE PREC. NOS IN ARRAY=DSAM*2
00140-000000 MODE:    0         ; FFR-SEQU
00141-177000 BLENG:   177000    ; LENGTH OF TAPE BUFFER
00142-072000 BTAPE:   72000     ; TAPE BUFFER
00143-040000 PULSE:   40000     ; WHERE THE PULSE SHAPE IS
00144-000400 PLENG:   400       ; LENGTH OF PULSE BUFFER
00145-000000 MICRO:   0
00146-000000 NOVA:    0
00147-000000 MNGN:    0
00150-000005 MXGN:    5
00151-020000 MAX1:    20000
00152-002000 MIN1:    2000
00153-000010 RXP:     10        ; RX PULSE=BIT4
0126  SABRE
00154-000020 TXP:     20        ; TX PULSE=BIT5 (BIT1-3=AMPGN)
00155-000040 SXP:     40        ; SENSE INHIBIT = BIT 6
00156-000054 PNUM:    44.       ; ELEMENTS IN PARAMETER ARRAY
                      ;*****************************************
                      ;* MISCELLANEOUS VARIABLES USED          *
                      ;*****************************************
00157-000050 WR1:     50        ; TAPE WRITE COMMAND
00160-000060 EOF1:    60
00161-000010 REWD:    10        ; REWIND
00162-000000 RBLK:    0         ; READ
00163-000000 EOF:     0         ; END OF FILE FLAG
00164-000014 YRS:     14        ; CLOCK END ADDRESS
00165-000016 SS:      16        ; CLOCK START/STOP ADDRESS
00166-000000 CUNT:    0         ; COUNTER
00167-000000 CUNT1:   0         ; COUNTER 1
00170-000000 TYPE:    0         ; TYPE OF PLOT
00171-000000 ADX3:    0
00172-000000 MADX:    0
00173-000000 NWDS:    0
00174-000032-PNAVE:   NAVE
00175-032563' MCSTK:  LASTLOC+24000
00176-001726'.MON:    BEGIN
00177-000200-KAPOW:   .+1
00200-004052'FAILP              ; POWER FAIL ADDRESS FOR NOVA 3
00201-000020 SL:      20
00202-101030 LC:      101030
00203-000000 ERRN:    0
00204-177577 MSK0:    177577
00205-006443'INT1:    INT
00206-000000 EOM:     0
00207-000000 LEM:     0
00210-040026 SYN:     40026
00211-000026 SYN1:    26
00212-000000 SV0:     0
00213-000000 SV1:     0
00214-000000 SV2:     0
00215-000000 SV3:     0
00216-000000 SINK:    0
00217-000000 JOBT:    0
00220-000000 JOBR:    0
00221-055446"ENDBF:   B*2+100
00222-000000 CBUF:    0
00223-000004 NSYNC:   4
00224-000004 CNT1:    4
```

```
                        ;*****************************************
                        ;* LASTLY ZERO PAGE                      *
                        ;*****************************************
   06563'006564'LASTLOC:        .+1
         035000         .BLK 35000
                        .ZREL
   00000-000000 WB:     0
   00001-000000 WE:     0
   00002-000000 START:  0
   00003-000000 GSTART: 0
   00004-000000 STATE:  0
   00005-000000 ERR:    0
   00006-006126'DL:     XTXFF
   00007-026263'TABLE:  LASTLOC+17500
   00010-000000 TPOINT: 0
         026563'B=LASTLOC+20000
   00011-055346"STORE:  B*2
   00012-001760'NONZRO: OPSTK
→ 0125  SABRE
                        ;*****************************************
                        ;* THE FAMOUS RADAR PARAMERERS           *
                        ;*****************************************
   00013-003675 YEAR:   1981.
   00014-000014-TBUF:   .
         000014         .BLK 14
   00031-000002 INTTIM: 2
   00032-000000 NAVE:   0
   00033-000055 TXPOW:  45.       ; TX POWER ?
   00034-000012 AWID1:  10.       ; PULSE WIDTH
   00035-177324 DOUB:   177324    ; DOUBLE PULSE SEPARATION
   00036-000062 NRANG:  50.       ; NO. OF RANGES
   00037-171270 SDLY:   -3400.    ; PRE-RUN WAIT (3.4 MSEC)
   00040-177634 GDLY:   177634    ; INTER-SAMPLE GAP
   00041-000010 NBEAM:  10        ; NO. OF BEAMS=8
   00042-000005 CCWAN:  5.        ; CCW BEAM ???
   00043-000000 AMPGN:  0         ; AMP GAIN
   00044-000144 AMPRT:  100.      ; AMP RISE TIME
   00045-000045-PLOC:   .
                        .TXT "NORMAL OPERATION FROM WICK "
   00046-047117
   00047-051115
   00050-040514
   00051-020117
   00052-050105
   00053-051101
   00054-052111
   00055-047516
   00056-020106
   00057-051117
   00060-046440
   00061-053511
   00062-041513
   00063-020000
   00064-000003 PGAP:   3         ; SAMPLES IN GEP
   00065-000065-DVALS:  .         ; ADDRESS OF ERROR CODES
```

```
        040000          .LOC 40000
 40000  000001  1
 40001  000001  1
 40002  000001  1
 40003  000001  1
 40004  000001  1
 40005  000001  1
 40006  000001  1
 40007  000001  1
 40010  000001  1
 40011  000001  1
 40012  000001  1
 40013  000001  1
 40014  000001  1
 40015  000001  1
 40016  000001  1
 40017  000001  1
 40020  000001  1
 40021  000001  1
 40022  000001  1
 40023  000001  1
 40024  000001  1
 40025  000001  1
 40026  000001  1
 40027  000001  1
 40030  000001  1
 40031  000001  1
 40032  000001  1
 40033  000100  100
 40034  000140  140
 40035  000200  200
 40036  000220  220
 40037  000240  240
 40040  000260  260
 40041  000300  300
 40042  000310  310
 40043  000320  320
 40044  000324  324
 40045  000330  330
 40046  000334  334
 40047  000340  340
 40050  000344  344
 40051  000350  350
 40052  000353  353
 40053  000356  356
 40054  000361  361
 40055  000364  364
 40056  000367  367
 40057  000372  372
 40060  000375  375
 40061  000377  377
 40062  000377  377
 40063  000377  377
 40064  000377  377
 40065  000377  377
0128  SABRE
```

```
                        ;*****************************************
                        ;* CLOCK BUFFER FOR CURRENT TIME         *
                        ;*****************************************
00225-000225-EBUF:      .
      000005            .BLK 5
                        ;*****************************************
                        ;* AUTO INCREMENT LOCATIONS              *
                        ;*****************************************
      000021            .LOC 21 ; OK SO THE AUTO INCS OK
00021 000000 PAUTO:     0
00022 000000 D1AUT:     0
00023 000000 D2AUT:     0
00024 006563' DP:       LASTLOC
00025 005457' MMAIN-1   ; ADRESS OF MICRONOVA JMP @ ON POWER FAIL
+ 0127  SABRE

                        ;*****************************************
                        ;* NOW WE CAN DEFINE THEE PULSE SHAPE    *
                        ;*****************************************
```

```
40066 000377 377
40067 000377 377
40070 000377 377
40071 000377 377
40072 000377 377
40073 000377 377
40074 000377 377
40075 000377 377
40076 000377 377
40077 000377 377
40100 000377 377
40101 000377 377
40102 000377 377
40103 000377 377
40104 000377 377
40105 000377 377
40106 000377 377
40107 000377 377
40110 000377 377
40111 000377 377
40112 000377 377
40113 000377 377
40114 000377 377
40115 000377 377
40116 000377 377
40117 000377 377
40120 000377 377
40121 000377 377
40122 000377 377
40123 000377 377
40124 000377 377
40125 000377 377
40126 000377 377
40127 000377 377
40130 000377 377
40131 000377 377
40132 000377 377
40133 000377 377
40134 000377 377
40135 000377 377
40136 000377 377
40137 000377 377
40140 000377 377
40141 000377 377
40142 000377 377
40143 000377 377
40144 000377 377
40145 000377 377
40146 000377 377
40147 000377 377
40150 000377 377
40151 000377 377
40152 000377 377
40153 000377 377
40154 000377 377
40155 000377 377
40156 000377 377
40157 000377 377
40160 000375 375
0129  SABRE
```

```
40161 000373 373
40162 000371 371
40163 000367 367
40164 000365 365
40165 000363 363
40166 000361 361
40167 000357 357
40170 000355 355
40171 000353 353
40172 000351 351
40173 000347 347
40174 000345 345
40175 000343 343
40176 000341 341
40177 000337 337
40200 000335 335
40201 000333 333
40202 000332 332
40203 000326 326
40204 000323 323
40205 000320 320
40206 000315 315
40207 000313 313
40210 000311 311
40211 000306 306
40212 000303 303
40213 000301 301
40214 000276 276
40215 000273 273
40216 000271 271
40217 000266 266
40220 000263 263
40221 000260 260
40222 000255 255
40223 000252 252
40224 000247 247
40225 000244 244
40226 000240 240
40227 000236 236
40230 000232 232
40231 000226 226
40232 000220 220
40233 000210 210
40234 000200 200
40235 000170 170
40236 000160 160
40237 000150 150
40240 000140 140
40241 000130 130
40242 000120 120
40243 000110 110
40244 000100 100
40245 000070 70
40246 000060 60
40247 000050 50
40250 000040 40
40251 000030 30
40252 000000 0
40253 000000 0
0130  SABRE
40254 000000 0
40255 000000 0
40256 000000 0
40257 000000 0
40260 000000 0
40261 000000 0
40262 000000 0
40263 000000 0
      000300          .BLK 300
→ 0131  SABRE
```

```
                    000201
  40765 040766'.DBST:
  40766 000377 377
  40767 000000 00
  40770 005227'ACT1
  40771 005243'ACT2
  40772 000377 377
  40773 000000 00
  40774 005227'ACT1
  40775 005243'ACT2
  40776 000377 377
  40777 000000 00
  41000 005227'ACT1
  41001 005243'ACT2
  41002 000377 377
  41003 000000 00
  41004 005227'ACT1
  41005 005243'ACT2
  41006 000377 377
  41007 000000 00
  41010 005227'ACT1
  41011 005243'ACT2
  41012 000377 377
  41013 000000 00
  41014 005227'ACT1
  41015 005243'ACT2
  41016 000377 377
  41017 000000 00
  41020 005227'ACT1
  41021 005243'ACT2
  41022 000377 377
  41023 000000 00
  41024 005227'ACT1
  41025 005243'ACT2
  41026 000377 377
  41027 000000 00
  41030 005227'ACT1
  41031 005243'ACT2
  41032 000377 377
  41033 000000 00
  41034 005227'ACT1
  41035 005243'ACT2
  41036 000377 377
  41037 000000 00
  41040 005227'ACT1
  41041 005243'ACT2
  41042 000377 377
  41043 000000 00
  41044 005227'ACT1
  41045 005243'ACT2
  41046 000377 377
  41047 000000 00
  41050 005227'ACT1
  41051 005243'ACT2
  41052 000377 377
 0132  SABRE
```

```
;*****************************************
;* ERROR CODE DATE BASE                  *
;*****************************************
.BLK 201
.+1
  41053 000000 00
  41054 005227'ACT1
  41055 005243'ACT2
  41056 000377 377
  41057 000000 00
  41060 005227'ACT1
  41061 005243'ACT2
  41062 000377 377
  41063 000000 00
  41064 005227'ACT1
  41065 005243'ACT2
  41066 000377 377
  41067 000000 00
  41070 005227'ACT1
  41071 005243'ACT2
  41072 000377 377
  41073 000000 00
  41074 005227'ACT1
  41075 005243'ACT2
  41076 000377 377
  41077 000000 00
  41100 005227'ACT1
  41101 005243'ACT2
  41102 000377 377
  41103 000000 00
  41104 005227'ACT1
  41105 005243'ACT2
  41106 000377 377
  41107 000000 00
  41110 005227'ACT1
  41111 005243'ACT2
  41112 000377 377
  41113 000000 00
  41114 005227'ACT1
  41115 005243'ACT2
  41116 000377 377
  41117 000000 00
  41120 005227'ACT1
  41121 005243'ACT2
  41122 000377 377
  41123 000000 00
  41124 005227'ACT1
  41125 005243'ACT2
  41126 000377 377
  41127 000000 00
  41130 005227'ACT1
  41131 005243'ACT2
  41132 000377 377
  41133 000000 00
  41134 005227'ACT1
  41135 005243'ACT2
  41136 000377 377
  41137 000000 00
  41140 005227'ACT1
  41141 005243'ACT2
  41142 000377 377
  41143 000000 00
  41144 005227'ACT1
  41145 005243'ACT2
 0133  SABRE
```

```
41146 000377 377
41147 000000 00
41150 005227' ACT1
41151 005243' ACT2
41152 000377 377
41153 000000 00
41154 005227' ACT1
41155 005243' ACT2
41156 000377 377
41157 000000 00
41160 005227' ACT1
41161 005243' ACT2
41162 000377 377
41163 000000 00
41164 005227' ACT1
41165 005243' ACT2
      001726'          .END BEGIN
```

```
                        ;
                        ;ROUTINE TO SEND INFO TO THE RX AND TX
                        ;REFORMATTING THE GAIN INFO TO THE REQUIRED VALUSE
                        ;
04561' 111000 SETGN:    MOV 0,2 ;DUPLICATE AMPGN VALUE
04562' 100000           COM 0,0          ;COMPLEMENT
04563' 024754           LDA 1,SEVEN      ;GET MASK
04564' 123400           AND 1,0          ;GET GAIN BITS 0-2
04565' 124000           COM 1,1          ;COMPLEMENT MASK
04566' 147400           AND 2,1          ;ADD IN REST
04567' 123000           ADD 1,0          ;REFORM THE WHOLE WORD
04570' 063044           DOC 0,44         ;OK SEND IT
04571' 001400           JMP 0,3          ;BYE BYE BLACKBIRD
→ 0090  SABRE
```

Appendix 5

Communications Software

```
                        ;*****************************************************
                        ;* O.K.  SO NOW WE HAVE A RESIDENT LOADER/DEBUGGER/
                        ;* COMPILER/ASSEMBLER/INTERPRETER
                        ;*  SO FOR NOW WE HAVE THE SITE PROGRAMS (AS IS)
                        ;*****************************************************
                        ;
                        ;
                XMSRI:  .TXT "(0)(4)MSRI"
01771'000004
01772'046523
01773'051111
01774'000000
01775'001705'BMSRI:     XLEFT
01776'075401 MSRI:      PSHA 3
01777'071401            PSHA 2
02000'020131-           LDA 0,POWER             ; THE ADDRESS OF THE POWER ARRAY
02001'100400            NEG 0,0                 ; ARRANGE.POWER ADDRESS-1
02002'100000            COM 0,0                 ; FOR FIRST AUTO INC
02003'040021            STA 0,PAUTO             ; STORE AT POWER AUTO INC ADDRES
02004'020135-           LDA 0,PSAM              ; LOAD NO. OF POWER SAMPLES
02005'040573            STA 0,COUNT             ; STORE FOR LATER
02006'034133-           LDA 3,PARAY             ; ADDRESS OF POWER ARRAY
02007'030564 PPLUP:     LDA 2,MASK              ; 177400
02010'026021            LDA 1,@PAUTO            ; GET FIRST WORD
02011'133400            AND 1,2                 ; GET X-
02012'146700            SUBS 2,1                ; GET Y-
02013'151132            MOVZL# 2,2,SZC          ; TEST X < 0
02014'150400            NEG 2,2                 ; YES SO MAKE +VE
02015'125132            MOVZL# 1,1,SZC          ; TEST Y < 0
02016'120701            NEGS 1,0,SKP            ; YES SO MAKE +VE AND IN FORM -Y
02017'121300            MOVS 1,0                ; NO BUT GENERATE -Y ANYWAY
02020'145000            MOV 2,1                 ; MOVE X- TO ACC 1
02021'073301            MUL                     ; ACC0=X*X ACC1= -Y
02022'131000            MOV 1,2                 ; GET READY FOR, Y*Y
02023'073301            MUL                     ; ACC1=Y*Y+X*X
02024'031400            LDA 2,0,3               ; GET CONTENTS OF CURRENT PARAY
02025'147022            ADDZ 2,1,SZC            ; ADD IN CURRENT AND CHECK OVERF
02026'011401            ISZ 1,3                 ; YES SO INC HIGH ORDER OF DOUBL
02027'045400            STA 1,0,3               ; RE-STORE NEW VALUE
02030'175400            INC 3,3                 ; INC ADDX POINTER
02031'175400            INC 3,3                 ; TWICE FOR DOUBLE PRECISION
02032'014546            DSZ COUNT               ; COUNT DOWN
02033'000754            JMP PPLUP               ; REPEAT LOOP
02034'071601            POPA 2                  ; RETURN
02035'075601            POPA 3                  ; O.K.
02036'001400            JMP 0,3         ; EXIT
                        ;
→ 0051  SABRE
```

```
                        ;*************************************************
                        ;* COMPLEX PRODUCT OF 2 DATA ARRAYS PGAP         *
                        ;* IS THE TIME SLICE BETWEEN THE DOUBLE PULSE    *
                        ;* THE RESULTANT ARRAY DARAY IS IN THE FORM      *
                        ;* REAL (LOW) REAL (HI) IMAG (LOW) IMAG (HI)     *
                        ;*************************************************
                XMCMP:  .TXT "(0)(4)MCMP"
02037'000004
02040'046503
02041'046520
02042'000000
02043'001771'BMCMP:     XMSRI
02044'075401 MCMP:      PSHA 3
02045'071401            PSHA 2                  ; SAVE ACCS FOR LATER
02046'024135-           LDA 1,PSAM              ; NO. OF DOPPLER SAMPLES
02047'044531            STA 1,COUNT             ; SAVE FOR LATER
02050'024132-           LDA 1,DOPPLER           ; ADDRESS OF DOPPLER ARRAY START
02051'124400            NEG 1,1                 ; GENERATE DOPPLER-1
02052'124000            COM 1,1                 ; O.K.
02053'044022            STA 1,D1AUT             ; AUTO INC LOCN 1
02054'020130-           LDA 0,PMGAP             ; PULSE GAP
02055'123000            ADD 1,0                 ; FORM OFFSET INTO ARRAY
02056'040023            STA 0,D2AUT             ; AUTO INC LOCN 2
02057'034134-           LDA 3,DARAY             ; OUTPUT ARRAY
02060'024513 MDLUP:     LDA 1,MASK              ; 177400
02061'131000            MOV 1,2                 ; ACC1=ACC2=177400
02062'022022            LDA 0,@D1AUT            ; GET FIRST X1Y1
02063'107400            AND 0,1                 ; GENERATE X1-
02064'122700            SUBS 1,0                ; GENERATE Y1-
02065'040510            STA 0,Y1                ; TEMPORARY LOCATION
02066'044506            STA 1,X1                ; DITTO
02067'022023            LDA 0,@D2AUT            ; GET FIRST X2Y2
02070'113400            AND 0,2                 ; GENERATE X2-
02071'142740            SUBOS 2,0               ; GENERATE Y2- CARRY=0
02072'050504            STA 2,X2                ; TEMPORARY LOCATION
02073'040504            STA 0,Y2                ; DITTO
02074'125112            MOVL# 1,1,SZC           ; X1 < 0 ?
02075'124460            NEGC 1,1                ; YEP SO NEGATE
02076'151112            MOVL# 2,2,SZC           ; X2 < 0 ?
02077'150460            NEGC 2,2                ; YEP SO NEGATE ALSO
02100'073301            MUL                     ; ACC0=X1*X2 ACC1=Y2- ACC2=X2-
02101'175402            INC 3,3,SZC             ; INC OUTPUT ADDRESS + TEST CARR
02102'100422            NEGZ 0,0,SZC            ; MAKE IT +VE
02103'101020            MOVZ 0,0                ; FORCE CARRY=0
02104'030471            LDA 2,Y1                ; RECOVER Y1-
02105'151112            MOVL# 2,2,SZC           ; Y1 < 0 ?
02106'150460            NEGC 2,2                ; YES SO MAKE +VE
02107'125112            MOVL# 1,1,SZC           ; Y2 < 0 ?
02110'124460            NEGC 1,1                ; YES SO MAKE +VE
02111'073301            MUL                     ; ACC0=Y1*Y2 ACC1=X1*X2
02112'175403            INC 3,3,SNC             ; TEST CARRY OF MUL AND INC OUTP
02113'107001            ADD 0,1,SKP             ; ACC1=X1*X2+Y1*Y2
02114'106400            SUB 0,1         ; DITTO
02115'125112            MOVL# 1,1,SZC           ; TEST CARRY OF ABOVE
02116'000405            JMP NEG1                ; CARRY=1 SO NEGATE
02117'021776            LDA 0,-2,3              ; GET REAL LOWER (-2) SINCE INC
02120'107022            ADDZ 0,1,SZC            ; ADD IN NEW VALUE
02121'011777            ISZ -1,3                ; OVERFLOW SO ADD IN +1 TO HIGH
0052  SABRE
02122'000404            JMP POS1                ; JMP +VE OK
02123'021776 NEG1:      LDA 0,-2,3              ; GET REAL LOWER
02124'107023            ADDZ 0,1,SNC            ; ADD IN
02125'015777            DSZ -1,3                ; OVER SO,SUBTRACT 1 TO HIGH ORD
```

```
02126'045776 POS1:   STA 1,-2,3              ; STORE RESULT
02127'101020         MOVZ 0,0                ; CARRY=0
02130'024445         LDA 1,Y1                ; GET Y1-
02131'125112         MOVL# 1,1,SZC           ; SIGN Y1-
02132'124460         NEGC 1,1                ; NEGATIVE SO NEGATE
02133'030443         LDA 2,X2                ; GET X2-
02134'151112         MOVL# 2,2,SZC           ; X2 < 0 ?
02135'150460         NEGC 2,2                ; YES SO NEGATE
02136'073301         MUL                     ; ACC0=Y1*X2
02137'175402         INC 3,3,SZC             ; INC OUTPUT ADDX AND TEST CARRY
02140'100422         NEGZ C,0,SZC            ; FORM ACC0 +VE
02141'101020         MOVZ 0,0                ; SET CARRY=0
02142'024432         LDA 1,X1                ; RECOVER X1-
02143'125112         MOVL# 1,1,SZC           ; TEST SIGN X1
02144'124460         NEGC 1,1                ; -VE
02145'030432         LDA 2,Y2                ; RECOVER Y2-
02146'151112         MOVL# 2,2,SZC           ; Y2 < 0 ?
02147'150460         NEGC 2,2                ; YES SO NEGATE
02150'073301         MUL                     ; ACC0=X1*Y2 ACC1=Y1*X2
02151'175402         INC 3,3,SZC             ; INC ADDX + TEST CARRY
02152'107021         ADD 0,1,SKP             ; FORM X1*Y2+Y1*X2
02153'103400         SUB 0,1                 ; DITTO
02154'125112         MOVL# 1,1,SZC           ; TEST CARRY OF RESULT
02155'000405         JMP NEG2                ; IF -VE NEGATE
02156'021776         LDA 0,-2,3              ; IMAG LOWER
02157'107022         ADDZ 0,1,SZC            ; OVER ?
02160'011777         ISZ -1,3                ; YES SO BUMP HIGH ORDER
02161'000404         JMP POS2
02162'021776 NEG2:   LDA 0,-2,3              ; IMAG LOWER
02163'107023         ADDZ 0,1,SNC            ; UNDER ?
02164'015777         DSZ -1,3                ; DECREMENT HIGH ORDER
02165'045776 POS2:   STA 1,-2,3              ; SAVE IT
02166'014412         DSZ COUNT               ; FINISHED ?
02167'000671         JMP MDLUP               ; NO SO CARRY ON
02170'071601         POPA 2                  ; RECOVER ACCS
02171'075601         POPA 3                  ; OK
02172'001400         JMP 0,3                 ; BYE
                     ;*****************************************************
                     ;* VARIABLES USED BY THESE BABIES                    *
                     ;*****************************************************
02173'177400 MASK:   177400
02174'000000 X1:     0
02175'000000 Y1:     0
02176'000000 X2:     0
02177'000000 Y2:     0
02200'000000 COUNT:  0
→ 0053  SABRE
```

```
                        ;*********************************************
                        ;* SIMPLE WAIT ROUTINE FOR N MILLI SECS      *
                        ;*********************************************
                XWAIT:  .TXT "(0)(4)WAIT"
02201'000004
02202'053501
02203'044524
02204'000000
02205'002037'BWAIT:     XMCMP
02206'075401 WAIT:      PSHA 3                  ; SAVE ACC3
02207'021400            LDA 0,0,3               ; GET SECS
02210'100400            NEG 0,0                 ; NEGATE IT
02211'105000            MOV 0,1                 ; SAVE FOR LATER
02212'020411 WLUP:      LDA 0,CWAIT             ; 1 MS WAIT
02213'100400            NEG 0,0                 ; NEGATE
02214'101404            INC 0,0,SZR             ; OVER ?
02215'000777            JMP .-1                 ; NO
02216'125404            INC 1,1,SZR             ; NO. OF MSECS UP ?
02217'000773            JMP WLUP                ; NO !!!
02220'075601            POPA 3
02221'001401            JMP 1,3                 ; LEAVE
                        ;*********************************************
                        ;* NEEDED FOR THIS ONE WE HAVE.....          *
                        ;*********************************************
02222'000275'.P030:     POP
02223'001000 CWAIT:             1000
→ 0054  SABRE

                        ;*********************************************
                        ;* AVERAGING FOR DATA ARRAY WHOSE ADDX       *
                        ;*  IS ON THE PROVERBIAL STACK !!!           *
                        ;*********************************************
                XAVE:   .TXT "(0)(4)AVER"
02224'000004
02225'040526
02226'042522
02227'047000
02230'002201'SAVE:      XWAIT
02231'075401 AVE:       PSHA 3                  ; SAVE 3
02232'023400            LDA 0,@0,3              ; GET ADDRESS
02233'120400            NEG 0,0                 ; FORM ADDX-1
02234'120020            COM 0,0                 ; OK
02235'040022            STA 0,D1AUT             ; AUTO INC 1
02236'040023            STA 0,D2AUT             ; AUTO INC 2
02237'023401            LDA 0,@1,3              ; GET COUNT
02240'040740            STA 0,COUNT             ; OK SAVE IT
02241'071401            PSHA 2
02242'000000-           LDA 2,NAVE              ; NUMBER OF AVERAGES
02243'026022 ALUP:      LDA 1,@D1AUT            ; LOWER HALF
02244'022022            LDA 0,@D1AUT            ; UPPER HALF
02245'101113            MOVL# 0,0,SNC           ; UPPER < 0 ?
02246'002405            JMP APOS                ; YES GO MAKE +VE
02247'174003            COM 3,3                 ; FLAG -VE FOR LATER
02250'100000            COM 0,0                 ; COMPLEMENT UPPER
02251'124405            NEG 1,1,SNR             ; NEGATE LOWER
02252'101400            INC 0,0                 ; BUMP UPPER BY 1 TO MAKE +VE DO
02253'073101 APOS:      DIV                     ; FORM AVERAGE
02254'175132            MOVZL# 3,3,SZC          ; NO. -VE ORIGINALY ?
02255'124401            NEG 1,1,SKP             ; YES GO NEGATE
02256'175132            MOVZL# 3,3,SZC          ; WAS IT AGAIN ?
02257'174000            COM 3,3                 ; RESET ACC3 +VE
02260'045023            STA 1,@D2AUT            ; STORE RESULT
02261'014717            DSZ COUNT               ; FINISHED ?
02262'000761            JMP ALUP                ; NO !!
02263'071601            POPA 2
02264'075601            POPA 3                  ; READY ?
02265'001402            JMP 2,3                 ; GO !!
→ 0055  SABRE
```

```
                        ;*****************************************
                        ;* GETS CURRENT TIME AND STORES          *
                        ;*****************************************
                XCTIM:  .TXT "(0)(4)CTIM"
02266'000004
02267'041524
02270'044515
02271'000000
02272'002224'BCTIM:     XAVE
02273'071401 CTIM:      PSHA 2
02274'075401            PSHA 3
02275'030164-           LDA 2,YRS               ; YEARS ADDRESS
02276'020014-REDO:      LDA 0,TBUF              ; GET TIME BUFFER
02277'040022            STA 0,D1AUT             ; AUTO INC ADDX
02300'034421            LDA 3,FTN               ; 15 FOR EMPTY BUS
02301'102520            SUBZL 0,0               ; GENERATE +1
02302'063040            DOC 0,40                ; SET READ MODE
02303'101400 REED1:     INC 0,0                 ; SET ADDX=2
02304'060140            NIOS 40                 ; SET IT GOING DONE SET BY IOPLS
02305'061340            DOAP 0,40               ; SEND ADDRESS
02306'063540            SKPBZ 40                ; WAIT TO READ
02307'000777            JMP .-1                 ; WAITING
02310'065440            DIB 1,40                ; READ DATA
02311'046022            STA 1,@D1AUT            ; STORE IT
02312'136435            SUBZ# 1,3,SNR           ; CHECK FOR 15
02313'000763            JMP REDO                ; RE-READ ALL THE BITS
02314'142434            SUBZ# 2,0,SZR           ; FINISHED ?
02315'000766            JMP REED1               ; NOT YET
02316'075601            POPA 3                  ; EXIT
02317'071601            POPA 2                  ; OK
02320'001400            JMP 0,3                 ; GO
                        ;*****************************************
                        ;* BITS NEDED FOR THE OLD CLOCK          *
                        ;*****************************************
02321'000017 FTN:       17
→ 0056  SABRE
```

```
                        ;*****************************************
                        ;* GRABS CURRENT TIME AND GETS END       *
                        ;* TIME                                  *
                        ;*****************************************
                XETIM:  .TXT "<0><4>ETIM"
    02322'000004
    02323'042524
    02324'044515
    02325'000000
    02326'002265'BETIM: XCTIM
    02327'075401 ETIM:  PSHA 3
    02330'071401        PSHA 2
    02331'004742        JSR CTIM                ; GET CURRENT TIME EXACTLY
    02332'020014-       LDA 0,TBUF              ; SET POINTERS
    02333'040022        STA 0,D1AUT             ; AUTO INC 1
    02334'020225-       LDA 0,EBUF              ; END BUFFER TIME
    02335'040023        STA 0,D2AUT             ; AUTO INC 2
    02336'024031-       LDA 1,INTTIM            ; INTEGRATION TIME (IN TENS OF S
    02337'022022        LDA 0,@D1AUT            ; GET  SECS
    02340'042023        STA 0,@D2AUT            ; NO CHANGE HERE
    02341'022022        LDA 0,@D1AUT            ; TENS OF SECS
    02342'107020        ADDZ 0,1                ; ADD TO INT TIME
    02343'034434        LDA 3,SIXSEC            ; 6
    02344'152400        SUB 2,2                 ; NO OF MINS TO ADD NEXT
    02345'136032        ADCZ# 1,3,SZC           ; >=6?
    02346'000403        JMP .+3                 ; BY-PASS
    02347'151400        INC 2,2                 ; INCREMENT MINS
    02350'166420        SUBZ 3,1                ; LESS 60 SECS
    02351'046023        STA 1,@D2AUT            ; STORE IT
    02352'026022        LDA 1,@D1AUT            ; GET MINS
    02353'147020        ADDZ 2,1                ; ADD IN INCREMENT
    02354'152400        SUB 2,2                 ; FLAG INC FOR HOURS
    02355'020423        LDA 0,TEN               ; 10
    02356'122434        SUBZ# 1,0,SZR           ; MINS=10
    02357'000403        JMP .+3                 ; BY-PASS
    02360'151400        INC 2,2                 ; FLAG INC
    02361'126400        SUB 1,1                 ; ZERO MINS
    02362'046023        STA 1,@D2AUT            ; STORE IT
    02363'026022        LDA 1,@D1AUT            ; TENS OF MINS
    02364'147020        ADDZ 2,1                ; ADD IN INC
    02365'152400        SUB 2,2                 ; INC HOURS
    02366'136434        SUBZ# 1,3,SZR           ; HOUR BOUNDARY ?
    02367'000403        JMP .+3                 ; NO SO BYPASS
    02370'151400        INC 2,2                 ; YES SO FLAG EOF
    02371'126400        SUB 1,1                 ; ZERO TENS OF MINS
    02372'046023        STA 1,@D2AUT            ; STORE IT
    02373'050163-       STA 2,EOF               ; SAVE POTENTIAL END OF FILE
    02374'071601        POPA 2                  ; LEAVE
    02375'075601        POPA 3                  ; BYE
    02376'001400        JMP 0,3                 ; AND GO
                        ;*****************************************
                        ;* BITS ANDA PIECES NEEDED HERE          *
                        ;*****************************************
    02377'000006 SIXSEC: 6
    02400'000012 TEN:    12
  → 0057  SABRE
```

```
                        ;*****************************************
                        ;* FLAGS CURRENT TIME PERIOD             *
                        ;*****************************************
                XTIUP:  .TXT "(0)(4)TIUP"
  02401'000004
  02402'052111
  02403'052520
  02404'000000
  02405'002322'BXTIP:   XETIM
  02406'075401 TIUP:    PSHA 3
  02407'004664          JSR CTIM          ; GET TIME NOW !!
  02410'020014-         LDA 0,TBUF        ; GET TIME BUFFER
  02411'040022          STA 0,D1AUT       ; AUTO INC 1
  02412'020225-         LDA 0,EBUF        ; EBND BUFFER
  02413'040023          STA 0,D2AUT       ; AUTO INC 2
  02414'020414          LDA 0,CLUP        ; COUNT LOOP=4
  02415'040166-         STA 0,CUNT        ; STORE IT
  02416'026022 NEWT:    LDA 1,@D1AUT      ; GET CURRENT SECS
  02417'022023          LDA 0,@D2AUT      ; GET END SECS
  02420'122434          SUBZ# 1,0,SZR     ; SAME ?
  02421'000405          JMP NOTUP         ; NO SO NOT YET
  02422'014166-         DSZ CUNT          ; COUNT UP ?
  02423'000773          JMP NEWT          ; NEXT TIME
  02424'075601          POPA 3            ; TIME IS UP
  02425'001401          JMP 1,3           ; SO LEAVE
  02426'075601 NOTUP:   POPA 3  ; GET READY
  02427'001400          JMP 0,3           ; LEAVE
                        ;*****************************************
                        ;* FEW THINGS NEEDED HERE OK             *
                        ;*****************************************
  02430'000004 CLUP:    4
  02431'000262'.PU31:   PUSH
→ 0058  SABRE
```

```
                        ;*****************************************
                        ;* GENERATES THE TAPE DATA ARRAY         *
                        ;*****************************************
                XTAPE:  .TXT "(0)(4)TAPE"
02432'000004
02433'052101
02434'050105
02435'000000
02436'002401'QTAPE:     XTIUP
02437'075401 TAPE:      PSHA 3
02440'071401            PSHA 2
02441'020142-           LDA 0,BTAPE       ; TAPE BUFFER OUTPUT ARRAY
02442'100400            NEG 0,0           ; ARRAY LOC-1
02443'100000            COM 0,0           ; DONE
02444'040022            STA 0,D1AUT       ; AUTO INC 1
02445'020014-           LDA 0,TBUF        ; PARAMETER ARRAY LOCATION
02446'024522            LDA 1,TOFF        ; OFFSET
02447'123020            ADDZ 1,0          ; SUM
02450'040023            STA 0,D2AUT       ; AUTO INC 2
02451'020156-           LDA 0,PNUM                ;GET NUMBER OF PARAMETER
02452'024535            LDA 1,SIXCP               ;NUMBER & TIME +NPARM
02453'123020            ADDZ 1,0                  ;ADD TO GET TOTAL
02454'101300            MOVS 0,0          ;*****************************
02455'042022            STA 0,@D1AUT      ; DUMMY
02456'020013-           LDA 0,YEAR        ; YEAR
02457'101300            MOVS 0,0          ;*****************************
02460'042022            STA 0,@D1AUT      ; STORE IT
02461'034014-           LDA 3,TBUF        ; TIME !!
02462'021412            LDA 0,12,3        ; MONTHS
02463'025413            LDA 1,13,3        ; X10
02464'030505            LDA 2,TPOLE       ; TEN POLE TUDOR
02465'073301            MUL               ; DECIMAL MONTHS IN ACC1
02466'034504            LDA 3,MTHTB       ; MONTH TABLE
02467'102520            SUBZL 0,0         ; +1
02470'152400            SUB 2,2           ; ACCUMULATED DAYS=0
02471'106435 CDAY:      SUBZ# 0,1,SNR     ; FINISHED ?
02472'000410            JMP CDFIN         ; DONE
02473'061401            PSHA 0            ; SAVE IT
02474'021400            LDA 0,0,3         ; GET-A-MONTH
02475'113020            ADDZ 0,2          ; ACCUMULATE
02476'175400            INC 3,3           ; INC POINTER
02477'061601            POPA 0            ; GET COUNTER
02500'101400            INC 0,0           ; +1
02501'000770            JMP CDAY          ; CARRY ON
02502'071401 CDFIN:     PSHA 2            ; SAVE FOR LATER
02503'034014-.          LDA 3,TBUF        ; TIME BUFFER
02504'021407            LDA 0,7,3         ; DAYS
02505'025410            LDA 1,10,3        ; X10
02506'030463            LDA 2,TPOLE       ; TEN POLE !
02507'073301            MUL               ; ACC1=DAYS OF CURRENT MONTH
02510'061601            POPA 0            ; GET PREVIOUS DAYS
02511'107020            ADDZ 0,1          ; DAYS TOTAL
02512'125300            MOVS 1,1          ;*****************************
02513'046022            STA 1,@D1AUT      ; AND STORE IT
02514'021405            LDA 0,5,3         ; HRS
02515'025406            LDA 1,6,3         ; X10
02516'073301            MUL               ; DONE
02517'125300            MOVS 1,1          ;*****************************
02520'046022            STA 1,@D1AUT      ; STORE IT
0059  SABRE
```

```
02521'021403          LDA 0,3,3       ; MINS
02522'025404          LDA 1,4,3       ; X10
02523'073301          MUL             ; DONE
02524'125300          MOVS 1,1        ;*************************
02525'046022          STA 1,@D1AUT    ; STORE IT
02526'021401          LDA 0,1,3       ; SECS
02527'025402          LDA 1,2,3       ; X10
02530'073301          MUL             ; DONE
02531'125300          MOVS 1,1        ;*************************
02532'046022          STA 1,@D1AUT    ; STORE
02533'020156-         LDA 0,PNUM      ; GET NO.
02534'100400          NEG 0,0         ; 2'S
02535'026023 L1:      LDA 1,@D2AUT    ; GET FIRST PARAMETER
02536'125300          MOVS 1,1        ;*************************
02537'046022          STA 1,@D1AUT    ; STORE IT
02540'101404          INC 0,0,SZR     ; FINISHED ?
02541'000774          JMP L1          ; LOOP
02542'020133-         LDA 0,PARAY     ; AVERAGED POWER ARRAY
02543'105000          MOV 0,1         ; SAVE IT
02544'020135-         LDA 0,PSAM      ; POWER SAMPLES
02545'004410          JSR X1ER        ; TRANSFER
02546'020134-         LDA 0,DARAY     ; DOPPLER ARRAY
02547'105000          MOV 0,1         ; SAVE IT
02550'020135-         LDA 0,PSAM      ; DOPPLER SAMPLES
02551'004404          JSR X1ER        ; TRANSFER
02552'071601          POPA 2
02553'075601          POPA 3          ; RETURN
02554'001400          JMP 0,3         ; GO
02555'100400 X1ER:    NEG 0,0         ; NO. OF TRANSFERS
02556'124400          NEG 1,1         ; POINTER-1
02557'124000          COM 1,1         ; DONE IT
02560'044023          STA 1,D2AUT     ; AUTO INC 2
02561'026023 L2:      LDA 1,@D2AUT    ; GET CHAR
02562'125300          MOVS 1,1        ;*************************
02563'046022          STA 1,@D1AUT    ; STORE IT
02564'101404          INC 0,0,SZR     ; DONE ?
02565'000774          JMP L2          ; NO !
02566'001400          JMP 0,3         ; YES SO RETURN
                      ;*****************************************
                      ;* BITS NEEDED FOR TAPER                 *
                      ;*****************************************
02567'000275'.PO32:   POP
02570'000014 TOFF:    14
02571'000012 TPOLE:   12
02572'002573'MTHTB:   .+1
02573'000037 31.
02574'000034 28.
02575'000037 31.
02576'000036 30.
02577'000037 31.
02600'000036 30.
02601'000037 31.
02602'000037 31.
02603'000036 30.
02604'000037 31.
02605'000036 30.
02606'000037 31.
02607'000006 SIXCP:   6         ;TIME +NPARM
→ 0060  SABRE
```

```
                        ;*****************************************
                        ;* ARCTANGENTS OF ARRAY OF COMPLEX      *
                        ;*  NUMBERS +180=77770 -180=100010      *
                        ;*****************************************
                XATAN:  .TXT "(0)(4)ATAN"
 02610'000004
 02611'040524
 02612'040516
 02613'000000
 02614'002432'BATAN:    XTAPE
 02615'075401 ATAN:     PSHA 3              ; SAVE 3
 02616'071401           PSHA 2              ; AND 2
 02617'020134-          LDA 0,DARAY         ; DOPPLER ARRAY POINTER
 02620'040022           STA 0,D1AUT         ; AUTO 1
 02621'040023           STA 0,D2AUT         ; AUTO 2
 02622'014022           DSZ D1AUT           ; -1
 02623'014023           DSZ D2AUT           ; -1
 02624'020136-          LDA 0,DSAM          ; DOPPLER SAMPLES
 02625'040166-          STA 0,CUNT          ; STORE IN COUNTER
 02626'022022 LUP1:     LDA 0,@D1AUT        ; READ REAL PART
 02627'026022           LDA 1,@D1AUT        ; READ IMAG PART
 02630'004407           JSR ATAN1           ; CALC ARCTAN
 02631'046023           STA 1,@D2AUT        ; RE-STORE IT
 02632'014166-          DSZ CUNT            ; DECREMENT COUNTER
 02633'000773           JMP LUP1            ; NOT FINISHED
 02634'071601           POPA 2              ; LEAVE
 02635'075601           POPA 3  ; !
 02636'001400           JMP 0,3
                        ;
                        ; ATAN SUBROUTINE D.G. SPECIAL
                        ;
 02637'075401 ATAN1:    PSHA 3              ; SAVE 3
 02640'071401           PSHA 2              ; SAVE 2
 02641'101132           MOVZL# 0,0,SZC      ; REAL < 0 ?
 02642'000411           JMP XNEG            ; YES !
 02643'125132           MOVZL# 1,1,SZC      ; IMAG < 0 ?
 02644'000403           JMP YNEG            ; YES !
 02645'004426           JSR ANGLE           ; CALC ANGLE
 02646'000422           JMP EX1 ; LEAVE
 02647'124400 YNEG:     NEG 1,1             ; 2'S OF IMAG PART
 02650'004423           JSR ANGLE           ; CALC ANGLE
 02651'124400           NEG 1,1             ; NEGATE AGAIN
 02652'000416           JMP EX1             ; EXIT
 02553'100400 XNEG:     NEG 0,0             ; NEGATE REAL PART
 02654'125112           MOVL# 1,1,SZC       ; IMAG NEGATIVE
 02655'000406           JMP XYNEG           ; IF BOTH NEG DO BOTH
 02656'004415           JSR ANGLE           ; CALC ANGLE
 02657'121000           MOV 1,0             ; MOVE ANGLE
 02660'024463           LDA 1,MXANG         ; MAX ANGLE=180
 02661'106400           SUB 0,1             ; 180-ANGLE
 02662'000406           JMP EX1             ; LEAVE
 02663'124400 XYNEG:    NEG 1,1             ; NEGATE IMAG PART
 02664'004407           JSR ANGLE           ; GET ANGLE
 02665'020456           LDA 0,MXANG         ; MAX ANGLE
 02666'100400           NEG 0,0             ; NEGATE MAXANG
 02667'107000           ADD 0,1             ; -180+ANGLE
 02670'071601 EX1:      POPA 2
 02671'075601           POPA 3
0061   SABRE
 02672'001400           JMP 0,3             ; LEAVE
```

```
                        ;
                        ; SUBROUTINE ANGLE
                        ;
02673'075401 ANGLE:     PSHA 3
02674'071401            PSHA 2              ; O.K.
02675'106414            SUB# 0,1,SZR        ; REAL=IMAG ?
02676'000403            JMP NEX0            ; NO !
02677'024446            LDA 1,ANG45         ; 45 DEGREES = 17776
02700'000440            JMP BYE1            ; LEAVE
02701'176440 NEX0:      SUBO 3,3            ; ACC3=CARRY=0
02702'054445            STA 3,FLG1          ; FLAG IMAG<REAL
02703'106512.           SUBL# 0,1,SZC       ; CHECK IMAG<REAL
02704'000404            JMP NEX1            ; YES !
02705'010442            ISZ FLG1            ; IMAG>REAL !
02706'131120            MOVZL 1,2           ; PREPARE TO DIVIDE
02707'000403            JMP NEX2            ;
02710'111120 NEX1:      MOVZL 0,2           ; NORMALISE DATA
02711'121000            MOV 1,0             ; FOR DIVISION
02712'126440 NEX2:      SUBO 1,1            ; ACC1=CARRY=0
02713'073101            DIV                 ; DIVIDE
02714'030434 STRT:      LDA 2,0100          ; 100
02715'155220            MOVZR 2,3           ; 40
02716'020434            LDA 0,TBL           ; TAN TABLE
02717'113000            ADD 0,2             ; POINT ROUGHLY HALF WAY DOWN
02720'021000 LUP:       LDA 0,0,2           ; GET TABLE VALUE OF ATAN
02721'122513            SUBL# 1,0,SNC       ; TAN()TABLE VALUE
02722'172401            SUB 3,2,SKP         ; LESS
02723'173000            ADD 3,2             ; GREATER
02724'175224            MOVZR 3,3,SZR       ; HALVE TABLE STEP
02725'000773            JMP LUP             ; NEXT COMPARISON
02726'024424            LDA 1,TBL           ; TABLE POINTER
02727'132400            SUB 1,2             ; 2*ANGLE IN DEGREES
02730'024416            LDA 1,ANG05         ; 0.5 DEGREES = 133 OCTAL
02731'102400            SUB 0,0             ; CLEAR ACC0
02732'073301            MUL                 ; CALC ANGLE
02733'014414            DSZ FLG1            ; ANGLE>45 ?
02734'000404            JMP BYE1            ; NO
02735'121000            MOV 1,0
02736'024406            LDA 1,ANG90         ; 90=37774
02737'106400            SUB 0,1             ; 90 DEGREES - ANGLE
02740'071601 BYE1:      POPA 2
02741'075601            POPA 3
02742'001400            JMP 0,3
                        ;*****************************************
                        ;* BITS NEEDED FOR ATAN ROUTINE          *
                        ;*****************************************
02743'077770 MXANG:     77770
02744'037774 ANG90:     37774
02745'017776 ANG45:     17776
02746'000133 ANG05:     133
02747'000000 FLG1:      0
02750'000100 0100:      100
02751'000275'.POP1:     POP
02752'002752'TBL:       TBL
→ 0062  SABRE
```

```
                        ;****************************************
                        ;* TEE TANGENT TABLE EVERY 0.5 DEGREES*
                        ;****************************************
 02753'000436 286.
 02754'001074 572.
 02755'001532 858.
 02756'002170 1144.
 02757'002627 1431.
 02760'003265 1717.
 02761'003724 2004.
 02762'004363 2291.
 02763'005023 2579.
 02764'005463 2867.
 02765'006123 3155.
 02766'006564 3444.
 02767'007225 3733.
 02770'007667 4023.
 02771'010332 4314.
 02772'010775 4605.
 02773'011441 4897.
 02774'012106 5190.
 02775'012553 5483.
 02776'013222 5778.
 02777'013671 6073.
 03000'014341 6369.
 03001'015013 6667.
 03002'015465 6965.
 03003'016140 7264.
 03004'016615 7565.
 03005'017273 7867.
 03006'017752 8170.
 03007'020432 8474.
 03010'021114 8780.
 03011'021577 9087.
 03012'022264 9396.
 03013'022752 9706.
 03014'023442 10018.
 03015'024133 10331.
 03016'024627 10647.
 03017'025324 10964.
 03020'026023 11283.
 03021'026523 11603.
 03022'027226 11926.
 03023'027733 12251.
 03024'030442 12578.
 03025'031153 12907.
 03026'031667 13239.
 03027'032405 13573.
 03030'033125 13909.
 03031'033650 14248.
 03032'034375 14589.
 03033'035125 14933.
 03034'035660 15280.
 03035'036415 15629.
 03036'037156 15982.
 03037'037721 16337.
 03040'040470 16696.
 03041'041241 17057.
0063  SABRE
```

```
 03042' 042017 17423.
 03043' 042577 17791.
 03044' 043363 18163.
 03045' 044153 18539.
 03046' 044746 18918.
 03047' 045545 19301.
 03050' 046350 19688.
 03051' 047160 20080.
 03052' 047773 20475.
 03053' 050613 20875.
 03054' 051437 21279.
 03055' 052270 21688.
 03056' 053126 22102.
 03057' 053770 22520.
 03060' 054540 22944.
 03061' 055515 23373.
 03062' 056377 23807.
 03063' 057266 24246.
 03064' 060164 24692.
 03065' 061067 25143.
 03066' 062000 25600.
 03067' 062720 26064.
 03070' 063646 26534.
 03071' 064603 27011.
 03072' 065547 27495.
 03073' 066522 27986.
 03074' 067504 28484.
 03075' 070476 28990.
 03076' 071500 29504.
 03077' 072511 30025.
 03100' 073534 30556.
 03101' 074567 31095.
 03102' 075633 31643.
 03103' 076710 32200.
 03104' 077777 32767.
 03105' 077777 32767.
 03106' 077777 32767.
 03107' 077777 32767.
 03110' 077777 32767.
 03111' 077777 32767.
 03112' 077777 32767.
→ 0064  SABRE
```

```
                        ;*************************************
                        ;* TAPE WRITE ROUNTINE FROM TBUF     *
                        ;*************************************
                XTWRIT: .TXT "(0)(4)TWRIT"
03145'000004
03146'052127
03147'051111
03150'052000
03151'003131'BTWRIT: XBLK
03152'075401 TWRIT:  PSHA 3
03153'060422         DIA 0,22          ;GET STATUS WORD
03154'024434         LDA 1,EOFF        ;GET EOF MASK
03155'107414         AND# 0,1,SZR      ;EOT ?
03156'002434         JMP @EOTL         ;EOT FOUND JMP TO ROUTINE FOR DEALING WI
03157'020142-        LDA 0,BTAPE       ; ADDRESS
03160'062022         DOB 0,22          ; LOADED
03161'020156-        LDA 0,PNUM        ; PARAMETER LENGTH
03162'024427         LDA 1,SIXPC               ;TIME+NPARM
03163'123020         ADDZ 1,0                  ;ADD TO GET TOTAL
03164'024135-        LDA 1,PSAM        ; POWER SAMPLES
03165'107020         ADDZ 0,1          ; OK
03166'020135-        LDA 0,PSAM        ; DOPPLER SAMPLES
03167'123020         ADDZ 1,0          ; OK
03170'100400         NEG 0,0           ; 2'S=LENGTH
03171'040141-        STA 0,BLENG       ; SAVE IT
03172'063022         DOC 0,22          ; LOADED
03173'020157-        LDA 0,WR1         ; WRITE COMMAND
03174'061122         DOAS 0,22         ; DONE !
03175'020163-        LDA 0,EOF         ; END OF FILE FLAG
03176'101005         MOV 0,0,SNR       ; EOF?
03177'000407         JMP .+7           ; NO
03200'102400         SUB 0,0           ; ZERO IT
03201'040163-        STA 0,EOF         ; DONE
03202'020160-        LDA 0,EOF1        ; COMMAND
03203'063522         SKPBZ 22          ; WAIT FOR TAPE
03204'000777         JMP .-1           ; OK
03205'061122         DOAS 0,22         ; WRITE EOF
03206'075601         POPA 3
03207'001400         JMP 0,3
03210'001000 EOFF:   1000
03211'000006 SIXPC:  6         ;NPARM+TIME
03212'003372'EOTL:   EOTR
→ 0067  SABRE
```

```
                        ;****************************************
                        ;* PULSE INITIALISATION ROUTINE      *
                        ;****************************************
                XPSET:  .TXT "(0)(4)PSET"
  03113'000004
  03114'050123
  03115'042524
  03116'000000
  03117'002610'BPSET:   XATAN
  03120'075401 PSET:    PSHA 3              ; SAVE 3 AS ALWAYS
  03121'020037-         LDA 0,SDLY          ; PRE-SAMPLE DELAY
  03122'062043          DOB 0,43            ; LOADED
  03123'020040-         LDA 0,GDLY          ; INTER-SAMPLE GAP DELAY
  03124'061043          DOA 0,43            ; LOADED
  03125'020140-         LDA 0,MODE          ; FFT-SEQUENTIAL SCAN
  03126'063043          DOC 0,43            ; LOADED
  03127'075601          POPA 3              ; LEAVE
  03130'001400          JMP 0,3             ; BYE
→ 0065  SABRE

                        ;****************************************
                        ;* GRAB A BLOK OF THE DATA           *
                        ;****************************************
                XBLK:   .TXT "(0)(4)BLOK"
  03131'000004
  03132'041114
  03133'047513
  03134'000000
  03135'003113'BBLK:    XPSET
  03136'023400 BLOK:    LDA 0,@0,3          ; ADDRESS
  03137'062041          DOB 0,41            ; LOADED
  03140'023401          LDA 0,@1,3          ; NO. OF WORDS
  03141'100400          NEG 0,0             ; 2'S
  03142'063041          DOC 0,41            ; LOADED
  03143'060141          NIOS 41             ; GO
  03144'001402          JMP 2,3             ; EXIT
→ 0066  SABRE
```

```
                        ;*****************************************
                        ;* RE-ALLOCATES ARRAY STORAGE FOR      *
                        ;*  THE AURORAL RADAR PROGRAMS         *
                        ;*****************************************
                XSTORE: .TXT "(0)(4)STOR"
03213'000004
03214'051524
03215'047522
03216'000000
03217'003145'BSTORE: XTWRIT
03220'102400 ST1:    SUB 0,0          ; CLEAR 0
03221'071401         PSHA 2           ;SAVE ACC 2
03222'024041-        LDA 1,NBEAM      ; BEAMS
03223'030036-        LDA 2,NRANG      ; RANGES
03224'073301         MUL              ; SAMPLES
03225'044135-        STA 1,PSAM       ; NO. OF POWER SAMPLES
03226'024064-        LDA 1,PGAP       ; SAMPLES IN DP GAP
03227'030041-        LDA 2,NBEAM      ; NO OF BEAMS
03230'102400         SUB 0,0          ; ZERO ACC0
03231'073301         MUL
03232'020135-        LDA 0,PSAM       ; POWER SAMPLES
03233'123020         ADDZ 1,0         ; ADD IN EXTRA NEEDED
03234'040136-        STA 0,DSAM       ; DOPPLER SAMPLES
03235'044130-        STA 1,PMGAP      ; STORE NBEAM*PGAP
03236'020131-        LDA 0,POWER      ; POWER ARRAY START
03237'024135-        LDA 1,PSAM       ; NO OF POWER SAMPLES
03240'107020         ADDZ 0,1         ; OFFSET TO DOPPLER ARRAY
03241'125400         INC 1,1          ; +1 TO CHECK
03242'044132-        STA 1,DOPPLER    ; START OF DOPPLER ARRAY
03243'020136-        LDA 0,DSAM       ; DOPPLER SAMPLES
03244'101400         INC 0,0          ; +1 TO CHECK
03245'107020         ADDZ 0,1         ; START OF POWER
03246'044133-        STA 1,PARAY      ; POWER ARRAY
03247'020135-        LDA 0,PSAM       ; POWER SAMPLES
03250'101120         MOVZL 0,0        ; *2 FOR DOUBLE PRECISION
03251'101400         INC 0,0          ; +1
03252'101400         INC 0,0          ; +2 TO CHECK
03253'107020         ADDZ 0,1         ; START OF DOPPLER
03254'044134-        STA 1,DARAY      ; DONE
03255'101120         MOVZL 0,0        ; *4 FOR QUAD PRECISION
03256'101400         INC 0,0          ; +1
03257'101400         INC 0,0          ; +2 TO CHECK
03260'107020         ADDZ 0,1         ; START OF TAPE
03261'044142-        STA 1,BTAPE      ; TAPE BUFFER
03262'020135-        LDA 0,PSAM       ; POWER SAMPLE NO.
03263'101120         MOVZL 0,0        ; *2
03264'040137-        STA 0,DSAM1      ; FOR AVERAGING DOPPLER ARRAY
03265'102620         SUBZR 0,0        ; 1XXXXX
03266'122532         SUBZL# 1,0,SZC   ; TOO MUCH ?
03267'002176-        JMP @.MON        ; MONITOR
03270'071601         POPA 2           ;RESTORE ACC 2
03271'001400         JMP 0,3          ; OK
→ 0068  SABRE
```

```
                        ;****************************************
                        ;* GAIN MODIFYING PROGRAM               *
                        ;****************************************
                XGNCK:  .TXT "(0)(4)GNCK"
 03272'000004
 03273'043516
 03274'041513
 03275'000000
 03276'003213'BGNCK:    XSTORE
 03277'075401 GNCK:     PSHA 3          ; SAVE ACC3
 03300'071401           PSHA 2
 03301'020133-          LDA 0,PARAY     ; POWER ARRAY START ADDRESS
 03302'100400           NEG 0,0         ; ADDX=ADDX-1
 03303'100000           COM 0,0         ; DONE
 03304'040022           STA 0,D1AUT     ; AUTO INC 1
 03305'020135-          LDA 0,PSAM      ; POWER SAMPLES
 03306'040166-          STA 0,CUNT      ; COUNTER
 03307'030151-          LDA 2,MAX1      ; MAX
 03310'020152-          LDA 0,MIN1      ; MIN
 03311'101020           MOVZ 0,0        ; CARRY=0
 03312'025022 GNLUP:    LDA 1,@D1AUT    ; GET FIRST VALUE
 03313'132512           SUBL# 1,2,SZC   ; )MAX?
 03314'000422           JMP DNGN        ; YES SO DOWN GAIN
 03315'106513           SUBL# 0,1,SNC   ; (MIN?
 03316'101040           MOVO 0,0        ; NO SO SET CARRY
 03317'014166-          DSZ CUNT        ; COUNT=COUNT-1
 03320'000772           JMP GNLUP       ; REPEAT
 03321'101002           MOV 0,0,SZC     ; CHECK SOMEWHERE GAIN (MIN
 03322'000411           JMP EXTX        ; EXIT
 03323'030043-          LDA 2,AMPGN     ; AMPGAIN
 03324'034423           LDA 3,TXM       ; MASK TX CONTROL
 03325'173420           ANDZ 3,2        ; OK
 03326'020150-          LDA 0,MXGN      ; MIN ATTENUATION
 03327'142435           SUBZ# 2,0,SNR   ; REACHED IT ?
 03330'000403           JMP EXTX        ; YES
 03331'010043-          ISZ AMPGN       ; +1 UP THE GAIN
 03332'101000           MOV 0,0         ; JUST IN CASE
 03333'071601 EXTX:     POPA 2          ; RETURN
 03334'075501           POPA 3
 03335'001400           JMP 0,3         ; EXIT
 03336'030043-DNGN:     LDA 2,AMPGN     ; GET AMP GAIN
 03337'034410           LDA 3,TXM       ; MASK TX CONTROL
 03340'173420           ANDZ 3,2        ; OK
 03341'020147-          LDA 0,MNGN      ; CHECK IF GAIN =0
 03342'142435           SUBZ# 2,0,SNR   ; WELL ?
 03343'000770           JMP EXTX        ; ATTN=MAX
 03344'014043-          DSZ AMPGN       ; -1 DOWN THE GAIN
 03345'101000           MOV 0,0         ;DUMMY
 03346'000765           JMP EXTX        ; EXIT
 03347'000007 TXM:      7
→ 0069  SABRE
```

```
                        ;*****************************************
                        ;* SET UP THE PULSE SHAPES               *
                        ;*****************************************
                XPSEQ:  .TXT "(0)(4)PSEQ"
03350'000004
03351'050123
03352'042521
03353'000000
03354'003272'BPSEQ:     XSNCK
03355'075401 PSEQ:      PSHA 3
03356'260244            NIOC 44             ; SET ADDRESS COUNTER=0
03357'034143-           LDA 3,PULSE         ; START OF PULSE SHAPE
03360'024144-           LDA 1,PLENG         ; LENGTH OF BUFFER
03361'124400            NEG 1,1             ; 2'S
03362'021400            LDA 0,0,3           ; GET A VALUE
03363'061344            DOAP 0,44           ; SEND IT + INC ADDRESS
03364'175400            INC 3,3             ; FOR NEXT VALUE
03365'125404            INC 1,1,SZR         ; FINISHED ?
03366'000774            JMP .-4             ; NO SO GET NEXT
03367'060244            NIOC 44             ; RESET EVERYTHING
03370'075501            POPA 3              ; GET ADDRESS
03371'001400            JMP 0,3             ; LEAVE
```

```
                        ;*********************************
                        ; EXIT ROUTINE
                        ;*********************************
03372'000000 EQ[?:      .DO @,E14           ;TELL MICRO
03373'040145-           STA 0,NOVA
03374'003405            JSR @E15            ;WAIT 3 SECONS
03375'005670            5670                ;WHILE MICRO PICKS THIS UP
03376'002401            JMP @.+1
03377'003517'           TX9                 ;SWITCH TX OFF
03400'100004 E14:       100004
03401'002206'E15:       WAIT
+ 0071  SABRE
```

```
03467'061140 CLKRD:  DOAS 0,4C          ; SET ADDRESS
03470'020460         LDA 1,0,3          ; GET 1ST PARAMETER
03471'060340         DOBP 1,40          ; WRITE IT
03472'063640         SKPBZ 40
03473'000777         JMP .-1
03474'175400         INC 3,3            ; INC BUFFER ADDRESS
03475'101400         INC 0,0            ; INC CLOCK  ADDRESS
03476'142434         SUBZ# 2,0,SZR      ; FINISHED ?
03477'000770         JMP CLKRD          ; NO SO SET NEXT
03500'020165-        LDA 0,SS           ; START ADDRESS
03501'061140         DOAS 0,40          ; SEND IT
03502'102533         SUBZL 0,0          ; GENERATE +1
03503'060340         DOBP 0,40          ; START CLOCK
03504'063640         SKPBZ 40
03505'000777         JMP .-1
03506'101223 FIN1:   MOVZR 0,0,SNC      ;ENSURE STSTUS
03507'000403         JMP FIN3           ;NO
03510'020475         LDA 0,NST1         ; TELL MICRO
03511'040145-        STA 0,NOVA         ; DO IT
03512'071601 FIN3:   POPA 2
03513'075601         POPA 3             ; LEAVE
03514'001400         JMP 0,3
03515'101223 TXOFF:  MOVZR 0,0,SNC      ;IS IT TX OFF ?
03516'000770         JMP FIN1           ;YES
03517'034177-TX9:    LDA 3,KAPOW        ;GET POWER ON ADDESS
03520'020460         LDA 0,CYCLE        ;WANT TO CYCLE WHEN START
03521'041400         STA 0,0,3          ;STORE IT
03522'020043-        LDA 0,AMPGN        ;INTERFACE COMMAND
03523'024154-        LDA 1,TXP          ;TX PULSE
03524'122420         SUBZ 1,0           ;REMOVE
03525'063044         DOC 0,44           ;SEND IT
03526'060244         NIOC 44            ;STOP PULSES
03527'020145-OFFLOP: LDA 0,MICRO        ;GET COMMAND
03530'101133         MOVZL# 0,0,SNC             ;TEST IT
03531'000776         JMP OFFLOP                 ;NOT SET
03532'126400         SUB 1,1            ;0
03533'044145-        STA 1,MICRO        ;CLEAR REQUEST
03534'101223         MOVZR 0,0,SNC      ;REWIND TAPE
03535'000404         JMP NE11           ;NO
03536'061401         PSHA 0             ;STORE 0
03537'006435         JSR @ATREW         ;REWIND TAPE
03540'061601         POPA 0             ;RESTORE ACC 0
03541'101223 NE11:   MOVZR 0,0,SNC      ;READ A BLOCK ?
03542'000404         JMP NE22           ;NO
03543'061401         PSHA 0             ;SAVE ACC
03544'006431         JSR @ATREAD        ;READ IT
03545'061601         POPA 0             ;RESTORE
03546'101223 NE22:   MOVZR 0,0,SNC      ;RESET PULSE SEQ ?
03547'000404         JMP NEX3           ;NO
03550'061401         PSHA 0
03551'006425         JSR @APSEQ         ;DO IT
0073  SABRE
03552'061601         POPA 0             ;RESTORE ACC 0
03553'101223 NEX3:   MOVZR 0,0,SNC      ;RESTRAT PROGRAM ?
03554'000402         JMP NEX4           ;NO
03555'000413         JMP NEX6           ;YES BYE
03556'101223 NEX4:   MOVZR 0,0,SNC      ;RESET CLOCK ?
03557'000402         JMP NEX5           ;NO
03560'125000         MOV 1,1            ;DUMMY FOR THE MMOME
03561'101222 NEX5:   MOVZR 0,0,SZC      ;TX OFF ?
03562'000745         JMP OFFLOP         ;NO
03563'101223         MOVZR 0,0,SNC      ;ENQUIRE
03564'000743         JMP OFFLOP         ;NO UNKNOWN
03565'020421         LDA 0,NST2         ;TELL MICRO
03566'040145-        STA 0,NOVA         ;STORE
03567'000740         JMP OFFLOP         ;AND AGAIN
03570'034177-NEX6:   LDA 3,KAPOW
03571'020411         LDA 0,MRR1
03572'041400         STA 0,0,3          ;RESET POWER UP ADDRESS
03573'002404         JMP @.MR1          ;GO MAM GO
03574'004177'ATREW:  TREW
03575'004213'ATREAD: TREAD
03576'003355'APSEQ:  PSEQ
```

```
                        ;*****************************************
                        ;* THE ALLOCATION OF BITS IS AS BELOW *
                        ;*                                       *
                        ;* 15=REWIND THE TAPE UNIT               *
                        ;* 14=READ A BLOCK FROM THE TAPE         *
                        ;* 13=RESET THE PULSE SEQUENCE           *
                        ;* 12=RESTART THE PROGRAM FROM 0 SECS    *
                        ;* 11=SET THE CLOCK TO NEW TIME          *
                        ;* 10=TX OFF COMMAND                     *
                        ;* 9 = STATUS WORD                       *
                        ;*                                       *
                        ;*****************************************
                XCOMM:  .TXT "(0)(4)COMM"
03402' 000004
03403' 041517
03404' 046515
03405' 000000
03406' 003350' BCOMM:   XPSEQ
03407' 075401  COMM:    PSHA 3          ; SAVE ACC3
03410' 071401           PSHA 2
03411' 020145-          LDA 0,MICRO     ; COMMUNICATIONS WORD
03412' 126400           SUB 1,1         ; =0
03413' 044145-          STA 1,MICRO     ; CLEAR REQUEST
03414' 101223           MOVZR 0,0,SNC   ; REWIND ?
03415' 000411           JMP NOTREW      ; NO
03416' 061401           PSHA 0          ;SAVE ACC 0
03417' 006562           JSR @AWEOF
03420' 006561           JSR @AWEOF      ;DOUBLE EOF
03421' 024161-          LDA 1,REWD      ; REWIND
03422' 065122           DOAS 1,22       ; DONE
03423' 063522           SKPBZ 22        ; WAIT
03424' 000777           JMP .-1
03425' 061601           POPA 0          ;RESTORE ACC 0
03426' 101223 NOTREW:   MOVZR 0,0,SNC   ; READ A BLOCK?
03427' 000411           JMP NREAD       ; NO
03430' 024142-          LDA 1,BTAPE     ; TAPE BUFFER START
03431' 066022           DOB 1,22        ; SET IT
03432' 024141-          LDA 1,BLENG     ; BUFFER LENGTH
03433' 067022           DOC 1,22        ; SET IT
03434' 024162-          LDA 1,RBLK      ; READ BLOCK
03435' 065122           DOAS 1,22       ; DO IT
03436' 063522           SKPBZ 22        ; BUSY ?
03437' 000777           JMP .-1         ; YES
03440' 101223 NREAD:    MOVZR 0,0,SNC   ; RESET PULSE SHAPE
03441' 000404           JMP NPLSE       ; NO
03442' 061401           PSHA 0
03443' 024712           JSR PSEQ        ; DO IT
03444' 061601           POPA 0          ; RESTORE COMMAND WORD
03445' 101223 NPLSE:    MOVZR 0,0,SZC
03446' 002001           JMP 0.MR1       ; RESTART PROGRAM
03447' 101223           MOVZR 0,0,SNC   ; RESET CLOCK
03450' 000445           JMP TXOFF       ; TX OFF ?
03451' 102420           SUBZ 0,0        ; GENERATE 0
03452' 063240           DOCC 0,40       ; CLEAR INTERFACE
03453' 061140           DOAS 0,40       ; SEND ADDX+START
03454' 063340           DOBP 0,40       ; WRITE 0 @ 0
03455' 063540           SKPBZ 40        ; WAIT
03456' 000777           JMP .-1
00271  SABRE
03457' 024155-          LDA 1,SS        ; START/STOP ADDRESS
03460' 065140           DOAS 1,40       ; SEND ADDRESS
03461' 063740           DOBP 0,40       ; SET=STOP
03462' 063540           SKPBZ 40
03463' 000777           JMP .-1         ; WAIT
03464' 024155-          LDA 0,CBUF      ; SET TIME BUFFER
03465' 020154-          LDA 2,YRS       ; END ADDRESS
03466' 102520           SUBZL 0,0       ; GENERATE ADDRESS=1
```

```
03675'127120          ADDZL 1,1         ; PUT BACK
03676'175400 RINT:    INC 3,3           ; FOND N WHERE 2**N=S
03677'125224          MOVZR 1,1,SZR     ; KEEP DIVIDING BY 2
03700'000776          JMP .-2           ; OK
03701'020505          LDA 0,THREE       ; CHECK VALUE
03702'116423          SUBZ 0,3,SNC      ; SKIP IF N GE 3
03703'000460          JMP NSAM          ; NO POINTS
03704'175222          MOVZR 3,3,SZC     ; N BECOMES N/2
03705'040507          STA 0,FLAG        ; SAVE IT
03706'024507          LDA 1,P           ; P=NO OF POINTS 13*5=65
03707'174405          NEG 3,3,SNR       ; NEGATE FOR COUNTER
03710'000445          JMP FTEST         ; TEST
03711'125220          MOVZR 1,1         ; GAP SPACING
03712'175404          INC 3,3,SZR       ; CALCULATE GAP
03713'000776          JMP .-2           ; KEEP GOING
03714'034474 PLO:     LDA 3,FIVE1       ; DEPTH OF EACH SAMPLE PAD
03715'054502          STA 3,DEPTH       ; FOR LATER
03716'034500          LDA 3,W           ; WIDTH OF PAD=13
03717'030502          LDA 2,YC          ; GET Y VALUE
03720'121220          MOVZR 1,0         ; GAP SPACING/2
03721'116032 SIZE:    ADCZ# 0,3,SZC     ; KIP IF DX GE 13
03722'000411          JMP SIZ           ; OUTPUT VALUES
03723'014474          DSZ DEPTH         ; HELP
03724'000402          JMP .+2
03725'000427          JMP FTES          ; TO FLAG TEST
03726'151400          INC 2,2           ; ADD 2 TO Y VALUE
03727'151400          INC 2,2           ; OK
03730'151400          INC 2,2
03731'162400          SUB 3,0           ; DX-13
03732'000767          JMP SIZE
03733'071401 SIZ:     PSHA 2            ; SAVE CURRENT Y VALUE
03734'065401          PSHA 1            ; SAVE ACC1
03735'061401          PSHA 0            ; SAVE 0
03736'111000          MOV 0,2           ; SET UP
03737'024452          LDA 1,SIX1        ; 6
03740'102400          SUB 0,0           ; TO MULTIPLY
03741'073301          MUL               ; DONE
03742'121000          MOV 1,0           ; MOVE OFFSET TO ACC0
03743'030455          LDA 2,XC          ; XVALUE
03744'113020          ADDZ 0,2          ; X+OFFSET
03745'071042          DOA 2,42          ; OUTPUT
03746'061601          POPA 0            ; RESTORE 0
03747'065601          POPA 1            ; RESTORE 1
03750'071601          POPA 2            ; RESTORE 2
03751'072042          DOB 2,42                  ; OUTPUT Y VALUE
03752'123000          ADD 1,0           ; DX+GS
03753'000745          JMP SIZE
03754'125120 FTES:    MOVZL 1,1         ; FLAG TEST
03755'034437 FTEST:   LDA 3,FLAG        ; WAS SAMPLE UNEVEN POWER OF 2
03756'020430          LDA 0,THREE
03757'116404          SUB 0,3,SZR
03760'000403          JMP .+3
03761'054433          STA 3,FLAG
03762'000732          JMP PLO
03763'030436 NSAM:    LDA 2,YC          ; CURRENT Y
03764'034420          LDA 3,TONE        ; 21
03765'173020          ADDZ 3,2          ; Y INC
03766'050433          STA 2,YC          ; RESTORE Y CURRENT
03767'014435          DSZ BINS
0076  SABRE
03770'000663          JMP RLOOP
03771'014437          DSZ CBEAM
03772'000405          JMP .+5
03773'101240          MOVOR 0,0         ; GENERATE 1XXXXX
03774'061042          DOA 0,42          ; TO SWITCH BEAM OFF
03775'075501          POPA 3
03776'001402          JMP 2,3
03777'020424          LDA 0,YS          ; FOR BLUP
04000'024422          LDA 1,XS
04001'030404          LDA 2,SONE        ; 61
04002'147000          ADD 2,1           ; ADD IN
04003'000637          JMP BLUP
```

```
                        ;*****************************************
                        ;* BITS NEEDED BY THIS ONE               *
                        ;*****************************************
03577' 004355'.MR1:     MLUP1
03600' 003527'CYCLE:    OFFLOP
03601' 004256'AWEOF:    WEOF
03602' 004052'MRR1:     FAILP
03603' 077777 ADMASK:   77777
03604' 000000 GOMAN:    0
03605' 100010 NST1:     100010
03606' 100020 NST2:     100020
→ 0074   SABRE

                        ;*****************************************
                        ;* THE PLOTTING PROGRAM                  *
                        ;*****************************************
03607' 002206'.WAIT4:   WAIT
                XPLOT:  .TXT "(0)(4)PLOT"
03610' 000004
03611' 050114
03612' 047524
03613' 000000
03614' 003402'BPLOT:    XCOMM
03615' 075401 PLOT:     PSHA 3              ; SAVE ACC 3
03616' 102620           SUBZR 0,0           ; GENERATE 1XXXXX
03617' 062042           DOB 0,42            ; ERASE SCREEN
03620' 006767           JSR @.WAIT4         ; BLANKING DELAY
03621' 000010           10
03622' 102400           SUB 0,0             ; ZERO
03623' 062042           DOB 0,42            ; BLANK
03624' 006763           JSR @.WAIT4         ; WAIT
03625' 001000           1000                ; BLANKED
03626' 075601           POPA 3              ; GET POINTER BACK
03627' 075401           PSHA 3              ; AND RESTORE IT
03630' 020170-          LDA 0,TYPE          ; GET TYPE OF PLOT
03631' 024133-          LDA 1,PARAY         ; INTENSITY
03632' 101004           MOV 0,0,SZR         ; 0=INTENSITY 1=DOPPLER
03633' 024134-          LDA 1,DARAY         ; DOPPLER
03634' 030041-          LDA 2,NBEAM         ; NO. OF BEAMS IN SYSTEM
03635' 147000           ADD 2,1             ; ADDRESS IN ARRAY OF FIRST SAMPLE
03636' 044570           STA 1,INADD         ; SAVE START ADDRESS
03637' 050571           STA 2,CBEAM         ; CURRENT BEAM
03640' 025400           LDA 1,0,3           ; PATTERN START ADDRESS
03641' 021401           LDA 0,1,3           ; GET Y START ADDX
03642' 044556 BLUP:     STA 1,XC            ; SAVE X CURRENT
03643' 044557           STA 1,XS            ; SAVE X START
03644' 040555           STA 0,YC            ; SAVE Y CURRENT
03645' 040555           STA 0,YS            ; SAVE Y START
03646' 024036-          LDA 1,NRANG         ; NO. OF RANGES
03647' 044555           STA 1,BINS          ; TEMPORARY STORAGE
03650' 014556           DSZ INADD           ; DECREMENT START ADDRESS
03651' 024555           LDA 1,INADD         ; START ADDX-1
03652' 044555           STA 1,CADD          ; STORE AT CURRENT ADDRESS
03653' 102400 RLOOP:    SUB 0,0             ; ACC0=RANGE LOOP
03654' 034553           LDA 3,CADD          ; ACC3=START ADDRESS
03655' 025400           LDA 1,0,3           ; LOAD SAMPLE
03656' 030041-          LDA 2,NBEAM         ; NO. OF BEAMS
03657' 157000           ADD 2,3             ; ADDX OF NEXT SAMPLE
03660' 054547           STA 3,CADD          ; STORE AT CURRENT ADDRESS
03661' 176400           SUB 3,3             ; ACC3=0
03662' 030170-          LDA 2,TYPE          ; CHECK THE TYPE
03663' 151005           MOV 2,2,SNR         ; TEST FOR VELOCITY OR INTENSITY
03664' 000412           JMP RINT            ; INTENSITY SAMPLES
03665' 125112           MOVL# 1,1,SZC       ; CHECK IF VELOCITY IS +VE
03666' 124401           NEG 1,1,SKP         ; MAKE IT +VE
03667' 034522           LDA 3,SIX1          ; FOR THE PLOT
03670' 125300           MOVS 1,1            ; SWAP THE BYTES
03671' 030522           LDA 2,AMASK         ; TO MASK THE BOTTOM BYTE IE 377
03672' 147400           AND 2,1             ; MASK IT
03673' 125235           MOVZR# 1,1,SNR      ; TEST FOR OUT OF RANGE
03674' 000467           JMP NSAM            ; YES SO NO SAMPLES
0075   SABRE
```

```
                        ;*************************************
                        ;* THINGS NEEDED BY THIS ROUTINE     *
                        ;*************************************
   04004'000020 TONE:   16.
   04005'000144 SONE:   100.
   04006'000003 THREE:  3
   04007'000004 FOR:    4
   04010'000005 FIVE1:  5
   04011'000006 SIX1:   6
   04012'006400 SW:     6400
   04013'000377 AMASK:  377
   04014'000000 FLAG:   0
   04015'000101 P:      101
   04016'000015 W:      15
   04017'000000 DEPTH:  0
   04020'000000 XC:     0
   04021'000000 YC:     0
   04022'000000 XS:     0
   04023'000000 YS:     0
   04024'000000 BINS:   0
   04025'002400 SFIVE:  2400
   04026'000000 INADD:  0
   04027'000000 CADD:   0
   04030'000000 CBEAM:  0
→ 0077  SABRE

                        ;*************************************
                        ;* POWER FAIL PROTECT ROUTINE        *
                        ;* THIS CODE WILL BE EXECUTED AS     *
                        ;* POWER IS FAILING !!!!             *
                        ;*************************************
              XFAIL:    .TXT "(0)(4)FAIL"
   04031'000004
   04032'043101
   04033'044514
   04034'000000
   04035'003610'BFAIL:  XPLOT
   04036'060241 FAIL:   NIOC 41          ; STOP THE DATA CHANNEL
   04037'020411         LDA 0,URUN       ; ADDRESS OF THE POWER FAIL ENTRY
   04040'024177-        LDA 1,KAPOW      ; ADDRESS OF PFAIL
   04041'123020         ADDZ 1,0         ; FORM INSTRUCTION JMP @
   04042'176400         SUB 3,3          ; SUB 3,3 => LOCATION 0
   04043'041400         STA 0,0,3        ; PUT JMP @ 300 INTO LOC 0
   04044'102440         SUBO 0,0         ;GENERTE 0
   04045'040145-        STA 0,MICRO
   04046'040146-        STA 0,NOVA
   04047'000767  .      JMP FAIL                  ; CONTINUE UNTIL DIE
                        ;*************************************
                        ;* ONLY ONE THING NEEDED HEREE       *
                        ;*************************************
   04050'002000 URUN:   2000
   04051'004036'FAI1:   FAIL
→ 0078  SABRE
```

```
                        ;*********************************
                        ;FAILP THIS ROUTINE IS EXECURED ON POWER UP
                        ;*********************************.
   04052'020777 FAILP:  LDA 0,FAI1        ;POWER FAI;L ROUTINE
   04053'176520         SUBZL 3,3         ;+1
   04054'041400         STA 0,0,3         ;STORE IT
   04055'020175-        LDA 0,MCSTK       ;
   04056'061001         MTSP 0
   04057'102000         ADC 0,0           ;-1
   04060'062077         DOB 0,CPU         ;MASK ALL DEVICES
   04061'101000         MOV 0,0
   04062'060177         INTEN
   04063'102440         SUBO 0,0          ;0
   04064'040145-        STA 0,MICRO
   04065'020405         LDA 0,NM
   04066'040146-        STA 0,NOVA        ;PUT IN NOVA
   04067'004404         JSR SEAR          ;SEARCH TAPE
   04070'002401         JMP @.+1          ;START RUNNING
   04071'004355'        MLUP1
   04072'100001 NM:     100001
→ 0079  SABRE
```

```
                        ;*********************************************
                        ;SEARCH
                        ; THIS ROUTINE SEARCHES THROUGH THE TAPE LOOKING
                        ;FOR THE END OF DATA BEFORE THE POWER FAILEED
                        ; 4 CONDITIONS HAVE BEEN USED TO DEFINE WHERE THIS
                        ; IS
                        ;
                        ; 1   FIND INCONSITANT TIME
                        ; 2   BLANK TAPE
                        ; 3 EOT
                        ;4 10 CONSEQUTIVE ERRORS
                        ;  WHEN THIS CONDITION IS SATISFIED THE TAPE
                        ; IS BACK SPACED AND AN EOF IS WRITTEN
                        ;*********************************************
04073'054466 SEAR:      STA 3,SRET      ;RETURN ADDRESS
04074'102400            SUB 0,0         ;0
04075'040465            STA 0,EFLAG     ;CLEAR EFLAG
04076'063522            SKPBZ 22
04077'000777            JMP .-1         ;WAIT TILL TAPE IS READY
04100'004513 SE2:       JSR TREAD       ;READ FIRST BLOK
04101'004565            JSR ERTEP       ;CHECK FOR ERROR
04102'000776            JMP SE2         ;TRY AGAIN
04103'063077            HALT            ;HELP !!!!!!!!!!
04104'034465            LDA 3,PBLOCK    ;START OF PREVIOUS BLOCK
04105'020557            LDA 0,MIN5      ;-5
04106'040557            STA 0,TCOT      ;STORE COUNTER
04107'030142-           LDA 2,BTAPE     ;START OF BUFFER
04110'151400            INC 2,2         ;POINT TO YEAR
04111'021000 SELOP:     LDA 0,0,2       ;GET NUMBER
04112'101300            MOVS 0,0        ;SWAP BYTES
04113'041400            STA 0,0,3       ;STORE IT
04114'151400            INC 2,2
04115'175400            INC 3,3         ;INC POINTERS
04116'010547            ISZ TCOT        ;INCREMEMNT POINTERS
04117'000772            JMP SELOP       ;AGAIN
04120'020440 SE3:       LDA 0, MIN10    ;
04121'040441            STA 0,EFLAG     ;STORE -10 IN EFLAG
04122'004471            JSR TREAD       ;READ BLOCK
04123'004543            JSR ERTEP       ;ANY ERRORS
04124'000776            JMP .-2         ;TRY AGAIN
04125'000424            JMP BLT         ;NOT BERNIE !!!!!!
04126'034435            LDA 3,CBLOCK    ;ADD OF CURRENT BLOCK
04127'020535            LDA 0,MIN5      ;COUNTER -5
04130'040535            STA 0,TCOT      ;STORE IT
04131'030142-           LDA 2,BTAPE     ;BUFFER ADDRESS
04132'151400            INC 2,2         ;POINTER TO TIME
04133'021000 SELO1:     LDA 0,0,2       ;GET TIME
04134'101300            MOVS 0,0        ;SWAP BYTES
04135'041400            STA 0,0,3       ;STORE IT
04136'151400            INC 2,2
04137'175400            INC 3,3         ;INC POINTERS
04140'010525            ISZ TCOT        ;DONE
04141'000772            JMP SELO1       ;NO
04142'004547            JSR TIMCH       ;CHECK TIME
04143'000755            JMP SE3         ;NOT THERE YET
04144'102520            SUBZL 0,0       ;DONE
04145'004437            JSR BCKS        ;BACKSPACE 1 BLOCK
04146'004510 SE4:       JSR WEOF        ;WRITE EOF
0080  SABRE
04147'034412            LDA 3,SRET      ;RETURN ADDRESS
04150'001400            JMP 0,3         ;HOME
04151'020504 BLT:       LDA 0,BLANK     ;BLANK TAPE
04152'101124            MOVZL 0,0,SZR   ;BLANK TAPE FOUND
04153'000773            JMP SE4         ;YES
04154'022404            LDA 0,MIN10     ;NO OF ERORRS
04155'100000            COM 0,0         ;COMPLEMENT
04156'004426            JSR BCKS        ;BACKSPACE
04157'000767            JMP SE4         ;HOME
```

```
   04160'177766 MIN10:   -10.
   04161'000000 SRET:    0
   04162'000000 EFLAG:   0
   04163'004164'CBLOCK:  .+1
   04164'000000          0
   04165'000000          0
   04166'000000          0
   04167'000000          0
   04170'000000          0
   04171'004172'PBLOCK:  .+1
   04172'000000          0
   04173'000000          0
   04174'000000          0
   04175'000000          0
   04176'000000          0
→ 0081  SABRE

                         ;*******************************
                         ; REWIND THE TAPE
                         ;*******************************
   04177'020161-TREW:    LDA 0,REWD        ;COMMAND WORD
   04200'061122          DOAS 0,22         ;DO IT
   04201'063522          SKPBZ 22          ;WAIT
   04202'000777          JMP .-1           ;TILL DONE
   04203'001400          JMP 0,3           ;HOME
→ 0082  SABRE

                         ;*******************************
                         ;BACKSPACE NUMBER OF RECORD IN ACC 0
                         ;*******************************
   04204'100000 BCKS:    COM 0,0           ;2 COMPLEMENT
   04205'063022          DOC 0,22          ;SEND IT
   04206'020443          LDA 0,BSP         ;GET IT
   04207'061122          DOAS 0,22         ;DO IT
   04210'063522          SKPBZ 22
   04211'000777          JMP .-1           ;WAIT TILL DONE
   04212'001400          JMP 0,3           ;BYE
→ 0083  SABRE
```

```
                        ;*****************************
                        ;ERROR THIS ROUTINE CHECKS FOR ANY ERRORS
                        ;EITHER BLANK TAPE EOF EOT OR OTHER
                        ;
                        ;RETURNS     ;;;;;
                        ;
                        ; 0 ON EOF OR ERROR <10
                        ; 1 BLANK TAPE OR 10 ERRORS
                        ; 2 NO ERRORS
                        ;*****************************
  04266'024767 ERTEP:   LDA 1,BLANK       ;BLANK TAPE INDICATOR
  04267'125222          MOVZR 1,1,SZC     ;?
  04270'001401          JMP 1,3           ;YES
  04271'101113          MOVL# 0,0,SNC     ;ANY ERRORS
  04272'001402          JMP 2,3           ; NO
  04273'024414          LDA 1,EOTM        ;GET EOT MADK
  04274'107414          AND# 0,1,SZR      ; EOT
  04275'002413          JMP @EOT1         ;EOT ROUTINE
  04276'125220          MOVZR 1,1         ;EOF MASK
  04277'107414          AND# 0,1,SZR      ;EOF ?
  04300'001400          JMP 0,3           ;YES
  04301'125220          MOVZR 1,1         ;BOT MASK
  04302'107414          AND# 0,1,SZR      ; BOT ?
  04303'001402          JMP 2,3           ;YES NORMAL RETURN A ????
  04304'010656          ISZ EFLAG         ;INCREMEMNT COUNTER
  04305'001400          JMP 0,3           ;TRY AGAIN
  04306'001001          JMP 1,2           ;FOUND IT
  04307'001000 EOTM:    1000
  04310'003372'EOT1:    EOTR
→ 0086  SABRE

                        ;*****************************
                        ;TIME CHECK THIS ROUTINE CHECKS CURRENT TIME
                        ; WITH PREVIOUS TIME
                        ;IF CURRENT <PREVIOUS THEN END OF DATA
                        ;*****************************
  04311'054435 TIMCH:   STA 3,TIMRET      ;RETURN ADDRESS
  04312'034651          LDA 3,CBLOCK      ;CURRENT BLOCK
  04313'030656          LDA 2,PBLOCK      ;PREVIOUS BLOCK
  04314'020750          LDA 0,MIN5        ;COUNTER
  04315'040750          STA 0,TCOT        ;STORE IT
  04316'021400 TLOOP:   LDA 0,0,3         ;GET CURRENT TIME
  04317'025000          LDA 1,0,2         ;PREVIOUS TIME
  04320'106432          SUBZ# 0,1,SZC     ;IS CURRENT > PREVIOS
  04321'000402          JMP TL1           ;NO
  04322'000407          JMP TEXIT         ;YES
  04323'106414 TL1:     SUB# 0,1,SZR      ;CURRENT =PREVIOUS
  04324'000420          JMP FOUND         ;NO THEREFORE C<P
  04325'175400          INC 3,3
  04326'151400          INC 2,2           ;INCREMEMNTPOINTERS
  04327'010736          ISZ TCOT          ;DONE ?
  04330'000766          JMP TLOOP         ; NO
  04331'020733 TEXIT:   LDA 0,MIN5        ;-5
  04332'040733          STA 0,TCOT        ;COUNTER
  04333'034630          LDA 3,CBLOCK      ;CURRENT
  04334'030635          LDA 2,PBLOCK      ;ADDRESS OF PREVIOUS
  04335'021400 TLOP1:   LDA 0,0,3         ;GET CURRENT TIME
  04336'041000          STA 0,0,2         ;PUT IN PREVIOUS
  04337'175400          INC 3,3
  04340'151400          INC 2,2           ;INC POINTERS
  04341'010724          ISZ TCOT          ;DONE ?
  04342'000773          JMP TLOP1         ; NO
  04343'002403          JMP @TIMRET       ;HOME
  04344'034402 FOUND:   LDA 3,TIMRET      ;FOUND INCON TIME.
  04345'001401          JMP 1,3
  04346'000000 TIMRET:  0
  04347'100002 ND:      100002
→ 0087  SABRE
```

```
                        ;*********************************
                        ; ROUTINE READ THIS ROUTINE READ A BLOCK FROM
                        ;THE TAPE AND CHECKS FOR BLANK TAPE
                        ; STATUS WORD IS RETURNED IN ACC 0
                        ;*********************************
04213'054437 TREAD:     STA 3,TRRET       ;RETURN ADDRESS
04214'102440            SUBO 0,0          ;0
04215'040440            STA 0,BLANK       ;CLEAR BLANK TAPE INDICTOR
04216'024142-           LDA 1,BTAPE       ;BUFFER
04217'066022            DOB 1,22          ;SEND IT
04220'024434            LDA 1,BBL         ;NO OF WORDS
04221'067022            DOC 1,22          ;SEND IT
04222'024162-           LDA 1,RBLK        ;READ COMMAND
04223'065122            DOAS 1,22         ;SEND IT
04224'006427            JSR @WAI1         ;1 SECOND WAIT
04225'001750            1750
04226'063522            SKPBZ 22          ;DONE
04227'000404            JMP WAI2          ; NO
04230'060422 STATUS:    DIA 0,22          ;GET STATUS WORD
04231'034421 STAT1:     LDA 3,TRRET
04232'001400            JMP 0,3           ;HOME
04233'006420 WAI2:      JSR @WAI1         ;5SECONS
04234'011510            11610
04235'063522            SKPBZ 22          ;DONE ?
04236'000402            JMP WAI3          ; NO
04237'000771            JMP STATUS        ;DONE
04240'126000 WAI3:      ADC 1,1           ;-1
04241'067222            DOCC 1,22         ;STOP TAPE
04242'024407            LDA 1,BSP         ;BACK COMMAND
04243'065122            DOAS 1,22         ;DO IT
04244'063522            SKPBZ 22
04245'000777            JMP .-1           ;WAIT TILL DNE
04246'126520            SUBZL 1,1         ;1
04247'044406            STA 1,BLANK       ;SET BLANK TAPE MARKER
04250'000761            JMP STAT1         ;BYE
04251'000040 BSP:       40
04252'000000 TRRET:     0
04253'002206'WAI1:      WAIT
04254'176256 BBL:       -850.
04255'000000 BLANK:     0
→ 0084  SABRE

                        ;*********************************
                        ; WRITE EOF
                        ;*********************************
04256'020160-WEOF:      LDA 0,EOF1
04257'061122            DOAS 0,22         ;SEND EOF COMMAND
04260'063522            SKPBZ 22
04261'000777            JMP .-1           ;WAIT TILL DONE
04262'001400            JMP 0,3           ;BYE
04263'004036'.FAIL:     FAIL
04264'177773 MIN5:      -5
04265'000000 TCOT:      0
→ 0085  SABRE
```

```
04435' 024145-          LDA 1,MICRO       ; MICRONOVA STATUS
04436' 125132           MOVZL# 1,1,SZC    ; ENFORCED WAIT ?
04437' 006515           JSR @.COM1        ; PROCESSOR/PROCESSOR COMMUNICATIONS
04440' 126620           SUBZR 1,1         ; GENERATE 100000
04441' 020133-          LDA 0,PARAY       ; POWER START
04442' 106420           SUBZ 0,1          ; DIFFERENCE IS CLEARED
04443' 040022           STA 0,D1AUT       ; AUTO INC LOCN 1
04444' 102440           SUBO 0,0          ; ACC0=CARRY=0
04445' 124400           NEG 1,1           ; NEGATE COUNT
04446' 042022           STA 0,@D1AUT      ; CLEAR
04447' 125404           INC 1,1,SZR       ; DONE ?
04450' 000776           JMP .-2           ; NOT YET
04451' 040032-          STA 0,NAVE        ; SET COUNT=0
04452' 063541 MLUP2:    SKPBZ 41          ; DONE ?
04453' 000777           JMP .-1           ; WAIT
04454' 060244           NIOC 44           ; CLEAR PULSER
04455' 020035-          LDA 0,DOUB        ; DOUBLE PULSE
04456' 062044           DOB 0,44          ; CALL INTERFACE
04457' 060144           NIOS 44           ; START PULSE
04460' 006462           JSR @.BLOK        ; GET DATA IN
04461' 000132-          DOPPLER           ; DOPPLER DATA
04462' 000136-          DSAM              ; NO. OF DOPPLER SAMPLES
                        ; NOW WE TAKE TX SENSE LOW
04463' 020032-          LDA 0,NAVE        ;NUMBER OF AVERAGES
04464' 024474           LDA 1,NTEST       ;START AFTER  NUMBER
04465' 106414           SUB# 0,1,SZR      ;EQUAL
04466' 000406           JMP MLUP3         ;NO CONTINUE
04467' 020043-          LDA 0,AMPGN       ;GET IT
04470' 024155-          LDA 1,SXP         ;SENSE THING
04471' 122420           SUBZ 1,0          ;REMOVE SXP
04472' 040043-          STA 0,AMPGN       ;
04473' 004466           JSR SETGN         ; DO IT
                        ; NOW IT HAS BEEN DONE
04474' 006444 MLUP3:    JSR @.MSRI        ; DO POWER CALCULATIONS
04475' 063541           SKPBZ 41          ; FINISHED ?
04476' 000777           JMP .-1           ; WAIT FOR NOW !!
04477' 060244           NIOC 44           ; CLEAR PULSER
04500' 020436           LDA 0,MINU7             ;  GET -5 OVER COME PROBLEM
04501' 062044           DOB 0,44          ; TELL PULSER
04502' 060144           NIOS 44           ; START PULSE
04503' 006437           JSR @.BLOK        ; GET DATA IN
04504' 000131-          POWER             ; POWER DATA
04505' 000135-          PSAM              ; NO. OF POWER SAMPLES
04506' 006433           JSR @.MCMP        ; DOPPLER CALC
04507' 010032-          ISZ NAVE          ; INC AVERAGE COUNT
04510' 006436           JSR @.TIUP        ; TIME UP ?
04511' 000741           JMP MLUP2         ; NO !!
04512' 006432           JSR @.ETIM        ; GET NEXT END TIME
                        ; NOW WE TAKE SENSE HI
04513' 020155-          LDA 0,SXP         ; INCREMENT
04514' 024043-          LDA 1,AMPGN       ; AMPSIGN
04515' 122220           ADDZ 1,0          ; DONE
04516' 040043-          STA 0,AMPGN       ; DONE
04517' 004442           JSR SETGN         ;SET GAIN
                        ; OK DONE
04520' 006425           JSR @.AVE         ; YES !
04521' 000133-          PARAY             ; POWER ARRAY
04522' 000135-          PSAM              ; NO OF SAMPLES TAKEN
04523' 006422           JSR @.AVE         ; AVERAGE
0099   SAGKE
04524' 000134-          DARAY             ; DOPPLER ARRAY
04525' 000137-          DSAM              ; DOPPLER SAMPLES
04526' 006422           JSR @.ATAN        ; CALC ARC TANGENTS
04527' 006420           JSR @.TAPE        ; TAPE ARRAY PROG
04530' 007021           JSR @.TWRITE      ; WRITE TAPE
04531' 006420           JSR @.PLOT        ; PLOT IT
04532' 000000           SZC               ; X START
04533' 000000           SKP               ; Y START
04534' 007017           JSR @.GAIN        ; CHECK GAIN
04535' 001378           JMP RUN           ; AND AGAIN
```

```
                        ;*****************************************
                        ;* MAIN PROCEDURE RUN ROUTINE            *
                        ;*****************************************
                MRUN:   .TXT "<0><4>MRUN"
04350'000004
04351'046522
04352'052516
04353'000000
04354'004031'BRUN:      XFAIL
04355'020706 MLUP1:     LDA 0,.FAIL         ; POWER FAIL ROUTINE
04356'176520            SUBZL 3,3           ; GENERATE +1
04357'041400            STA 0,0,3           ; STORE ADDRESS FOR INTERRUPT
04360'020175-           LDA 0,MCSTK
04361'061001            MTSP 0
04362'102000            ADC 0,0             ; GET -1
04363'062077            DOB 0,CPU           ; MSKO ALL DEVICES
04364'101000            MOV 0,0             ; WAIT
04365'060177            INTEN               ; ENABLE ALL INTERRUPTS FOR PFAIL
04366'060241            NIOC 41             ; RESTART INTERFACE
04367'020760            LDA 0,ND            ;TELL MICRO
04370'040146-           STA 0,NOVA          ;STORE
04371'102440            SUBO 0,0            ;GENERATE 0
04372'004567            JSR SETGN           ;SEND TX COMMAND LOW SO IT WILL
                                            ;CORRECTLY RESTART IF TX PULSE HIGH
04373'006564            JSR @.WAI5
04374'011661            11661                       ;DELAY FOR VOLTAGE RISE
04375'006555            JSR @.ST1           ; ALLOCATE STORAGE
04376'006557            JSR @.PSEQ          ; PULSE SEQUENCE
04377'020153-           LDA 0,RXP           ; PULSE RX
04400'063044            DOC 0,44            ; OK
04401'020154-           LDA 0,TXP           ; PULSE TX
04402'024155-           LDA 1,SXP           ; SENSE INHIBIT
04403'123020            ADDZ 1,0            ; TX=1 SENSE=1
04404'040043-           STA 0,AMPGN         ; AMPGAIN=BITS1-3,RXP=BIT4,TXP=BIT5,SXP=
04405'004554            JSR SETGN           ;SEND IT
04406'006551            JSR @.WAI5          ; WAIT=65 SECCS
04407'000000            0
04410'006547            JSR @.WAI5          ; WAIT=65 SECS
04411'000000            0
04412'006545            JSR @.WAI5          ; WAIT=30 SECS
04413'100000            100000
04414'006527 MLUP4:     JSR @.CTIM          ; GET CURRENT TIME
04415'034014-           LDA 3,TBUF          ; GET TIME BUFFER
04416'021402            LDA 0,2,3           ; TENS OF SECS
04417'101004            MOV 0,0,SZR         ; =0 ?
04420'000774            JMP MLUP4                   ; NO SO WAIT
04421'021401            LDA 0,1,3           ; SECS
04422'101004            MOV 0,0,SZR         ; =0 ?
04423'000771            JMP MLUP4           ; NO SO WAIT
04424'006520            JSR @.ETIM          ; GET END TIME
04425'020037-           LDA 0,SDLY          ; PRE-SAMPLE DELAY
04426'062043            DOB 0,43            ; SEND TO SAMPLE INTERFACE
04427'020040-           LDA 0,GDLY          ; INTER SAMPLE GAP DELAY
04430'061043            DOA 0,43            ; SEND TO SAMPLE INTERFACE
04431'020140-           LDA 0,MODE          ; FFT-SEQUENTIAL
04432'063043            DOC 0,43            ; SEND TO SAMPLE INTERFACE
04433'020043-RUN:       LDA 0,AMPGN         ; RECEIVER GAIN
04434'004525            JSR SETGN           ;SET GAIN
0088  SABRE
```

```
;*********************************************
;* MAIN MODULE THINGS NEEDED O.K.            *
;*********************************************
04535' 177771 MINU7:  -7
04536' 000007 SEVEN:  7
04540' 001776'.MSRI:  MSRI
04541' 002044'.MCMP:  MCMP
04542' 003136'.BLOK:  BLOK
04543' 002273'.CTIM:  CTIM
04544' 002327'.ETIM:  ETIM
04545' 002231'.AVE:   AVE
04546' 002406'.TIUP:  TIUP
04547' 002437'.TAPE:  TAPE
04550' 002515'.ATAN:  ATAN
04551' 003152'.TWRITE:        TWRITE
04552' 003220'.ST1:   ST1
04553' 003277'.GNCK:  GNCK
04554' 003407'.COM1:  COMM
04555' 003355'.PSEQ:  PSEQ
04556' 003615'.PLOT:  PLOT
04557' 002206'.WAI5:  WAIT
04560' 000010 NTEST:  10
```

```
Cx          ;
          ; CALLED BY READL(DATA ARRAY, WORK ARRAY, ERROR)
          ;
          .TITLE TREADL
          .ENT TARY RARY
          .ENT DATL
          .ENT READL ECODE
          .EXTN CODE
          .ENT WORK DARAY
          .EXTD CRC
          .TXTM 1
          .NREL
READL:    LDA 0,@.TARY      ; ARRAY ADDRESS
          STA 0,DARAY       ; FOR CRC
          LDA 1,THR         ; GET +3
          ADDZ 1,0          ; FIRST FREE LOCATION IN ARRAY
          STA 3,SAV3        ; SAVE ACC3
          MOVZL 0,0         ; GENERATE BYTE POINTER
          LDA 1,NBYTE       ; NO. OF BYTES TO READ
          .SYSTM            ; CALL RDOS
          .RDS 3            ; GET LINE
          MOV 0,0,SKP             ; ERROR ?
          ;
          ; IF ERROR THE NEXT INSTRUCTION IS SKIPPED
          ;
          SUBZ 2,2          ; FORCE IER=0
          ;
          STA 2,ECODE       ; STORE ERROR CODE FOR LATER
          ;
          MOVZR# 1,1,SZC    ; ODD NO. OF BYTES ?
          INC 1,1           ; YES SO MAKE EVEN
          STA 1,DATL        ; STORE BYTE COUNT
          MOVZR 0,0         ; GET WORD POINTER
          MOVZR 1,3         ; GET NO. OF WORDS
          ADDZ 0,3          ; NEXT FREE LOCATION
          ;
          LDA 0,ENDIT       ; ETX
          STA 0,0,3         ; STORE ETX
          LDA 3,SAV3        ; GET 3
          JMP 0,3           ; RETURN
          ;
          ; DATA
          ;
SAV3:     0
.TARY:    TARY
THR:      3
NBYTE:    400.
ENDIT:    10003    ; .DLE.ETX.
DATL:     0
NEWL:     0
ECODE:    0
.CODE:    CODE
.CRC:     CRC
DARAY:    0
WORK:     .+1
          .BLK 1000
TARY:     .+1
          13026
          13026
          10002
          .BLK 1000
RARY:     .+1
          .BLK 1000
          .END
```

```
D           .TITLE FTXMT
           .ENT TXMT
           .EXTN DATL RFLG TFLG MFLG TDAT RDAT STAT CODE
           .EXTN TARY PART
           .ENT CONTR
           .TXTM.1
           .DUSR ULM=34
           .NREL
.PART:     PART
RTSON:     100003
SAV3:      0
TXMT:      STA 3,SAV3          ; SAVE IT BOY
           NIOC 35             ; CLEAR CRC
           LDA 0,RTSON         ; RTS='1'
           DOB 0,ULM           ; SET
           LDA 0,SYN           ; GET A SYNC CHAR
           STA 0,@.TDAT        ; STORE IT
           LDA 0,SL            ; SET LINE
           DOA 0,ULM           ; DONE
           SUBZL 0,0           ; +1
           DOC 0,ULM           ; LINE=ON
           JSR TCHAR+1         ; WAIT TILL SYNC CHAR SENT
           LDA 3,SAV3          ; RESTORE ACC 3
           LDA 2,@.TARY        ; ARRAY ADDRESS
           LDA 1,CONTR         ; CONTROL BLOCK
           MOV 1,1,SZR         ; CONTROL/DATA ?
           JMP DATAB           ; DATA BLOK
           LDA 1,CONL          ; CONTROL BLOCK LENGTH
           NEG 1,1             ; 2'S
           ;
           ;
NEXT:      LDA 0,0,2           ; GET A WORD
           LDA 3,MASK          ; 377
           MOVS 0,0            ; GET HI BYTE
           ANDZ 3,0            ; ISOLATED
           JSR TCHAR           ; OUTPUT THE BYTE
           LDA 3,MASK          ; 377
           LDA 0,0,2           ; GET WORD
           ANDZ 3,0            ; LOW BYTE
           JSR TCHAR           ; OUTPUT IT
           INC 2,2             ; UP POINTER
           INCZ 1,1,SZR        ; DONE YET ?
           JMP NEXT            ; SEND NEXT
           JMP FINISH          ; YES SO GO
           ;
           ; DATA BLOCK FORMAT
           ;
DATAB:     LDA 1,M3            ; THREE WORDS (-3)
           ;
           ; FIRST THREE WORDS TRANSPARENTLY
           :
```

```
NEXT0:  LDA 0,0,2       ; GET WORD
        LDA 3,MASK      ; 377
        MOVS 0,0        ; TOP BYTE
        ANDZ 3,0        ; GET BYTE
        JSR TCHAR       ; SEND IT
        LDA 3,MASK      ; 377
        LDA 0,0,2       ; GET WORD
        ANDZ 3,0        ; GET BOTTOM BYTE
        JSR TCHAR       ; OUTPUT IT
        INC 2,2         ; UP POINTER
        INC 1,1,SZR     ; DONE 3 ?
        JMP NEXT0       ; NOT YET
        ;
        ; NOW THE DATA PART OF THE BLOCK
        ;
TRAIL:  LDA 1,@.DATL    ; DATA LENGTH
        MOVZR 1,1       ; WORD POINTER
        NEG 1,1         ; O.K.
        SUBZ 0,0
        STA 0,@.PART
NEXT1:  LDA 0,0,2       ; GET A WORD
        LDA 3,MASK      ; 377
        MOVS 0,0        ; GET HI BYTE
        ANDZ 3,0        ; ISOLATE IT
        LDA 3,DLE       ; DLE ?
        SUBZ# 3,0,SNR   ; WELL
        JSR TCHAR       ; SEND IT
        JSR TCHAR       ; SEND IT
        LDA 0,0,2       ; GET WORD
        LDA 3,MASK      ; 377
        ANDZ 3,0        ; ISOLATE IT
        LDA 3,DLE       ; DLE ?
        SUBZ# 3,0,SNR   ; WELL ?
        JSR TCHAR       ; PUT 2 IN
        JSR TCHAR       ; SEND IT
        INC 2,2         ; BUMP POINTER
        INCZ 1,1,SZR    ; DONE YET ?
        JMP NEXT1       ; NEXT
        ;
        ; FINALLY THE TRAILER
        ;
        SUBZL 1,1       ; GENERATE +1
        NEGZ 1,1        ; 177776
NEXT2:  LDA 0,0,2       ; GET WORD
        LDA 3,MASK      ; 377
        MOVS 0,0        ; TOP
        ANDZ 3,0        ; GET IT
        JSR TCHAR       ; ZAP
        LDA 3,MASK      ; 377
        LDA 0,0,2
        ANDZ 3,0        ; GET BYTE
        JSR TCHAR       ; OK
        INC 2,2         ; BUMP POINTER
        INCZ 1,1,SZR    ; DONE ?
        JMP NEXT2       ; NO !
        ;
        ; THE HARDWARE CRC CHECK
```

```
        ;
CRC:    SUBZL 1,1           ; +1
        INC 1,1             ;
        INC 1,1
        NEGZ 1,1            ; 2'S OF NO. OF WORDS TO DO
NEXT3:  LDA 0,@.PART        ; GET CRC
        STA 0,TEMP          ; STORE IT FOR LATER
        LDA 3,MASK          ; 377 O.K.
        MOVS 0,0            ; SWAP
        ANDZ 3,0            ; AND IT
        LDA 3,DLE           ; DLE.CHAR
        SUBZ# 3,0,SNR       ; ZAP IT
        JSR TCHAR           ; O.K.
        JSR TCHAR           ; AT LEAST !!
        LDA 0,TEMP          ; GET WORD AGAIN
        LDA 3,MASK          ; 377
        ANDZ 3,0            ; GET-A-BYTE
        LDA 3,DLE           ; DLE ?
        SUBZ# 3,0,SNR       ; ?
        JSR TCHAR           ; EXTRA DLE !!
        JSR TCHAR           ; AT LEAT ONE
        INC 1,1,SZR         ; COUNTER
        JMP NEXT3           ; NEXT ONE PLEASE
        ;
        ; DONE NOW
        ;
FINISH: LDA 0,@.TFLG
        MOV 0,0,SNR
        JMP FINISH
        LDA 0,SL            ; SET LINE
        DOA 0,ULM           ; SET IT
        SUB 0,0             ; +0
        DOC 0,ULM           ; TURN SECTION OFF
        LDA 3,SAV3          ; GET 3 BACK
        JMP 0,3
        ;
        ; BASIC TX ROUTINE
        ;
TCHAR:  STA 0,@.TDAT            ; STORE CHAR
        STA 3,SAV33             ; SAVE 3
        LDA 3,@.TFLG            ; GONE YET ?
        MOV 3,3,SNR             ; ?
        JMP .-2                 ; NO SO WAIT
        SUBZ 3,3                ; CLEAR FLAG
        STA 3,@.TFLG            ; STORE IT
        LDA 3,SAV33             ; RETURN
        JMP 0,3             ; NO
        ;
        ; DATA REQUIRED
        ;
SAV33:  0
.TFLG:  TFLG
.TDAT:  TDAT
.DATL:  DATL
.TARY:  TARY
CONTR:  0
SL:     21
SYN:    26
DLE:    20
CONL:   12
MASK:   377
TEMP:   0
TEN:    12
M3:     -3
        .END
```

```
        .TITLE MAIN
        .EXTD SETUP TXMT RXMT READL TEOB TENQ
        .EXTD TACK TNAK TDUM
        .EXTN ERR1 STAT1 ECODE CONTR
        .EXTN TEMP8
        .EXTN DARAY TARY RARY
        .TXTM 1
        .NREL
MS2:    .+1*2
        .TXT "TYPE 1 FOR TRANSMIT 0 FOR RECEIVE (12)(15)"
MS3:    .+1*2
        .TXT "CONNECT PROBLEMS (12)(15)"
MS4:    .+1*2
        .TXT "FINISHED O.K. (12)(15)"
TOUT:   .+1*2
        .TXT "$TTO"
TIN:    .+1*2
        .TXT "$TTI"
FILE:   .+1*2
        .TXT "INPUT"
FOUT:   .+1*2
        .TXT "OUTPUT"
START:  LDA 0,TOUT      ; OPEN $TTO
        SUB 1,1         ; MASK=0
        .SYSTM          ; CALL RDOS
        .OPEN 1         ; OPEN 1
        JMP @.ERROR     ; ERROR
        LDA 0,TIN       ; OPEN $TTI
        SUB 1,1         ;
        .SYSTM
        .OPEN 2
        JMP @.ERROR
        LDA 0,FOUT      ; OPEN TX FILE
        SUB 1,1
        .SYSTM
        .OPEN 3
        JMP @.ERROR
        LDA 0,FILE      ; OPEN RX FILE
        SUB 1,1
        .SYSTM
        .OPEN 4
        JMP @.ERROR
        ;
        ;
MESS1:  LDA 0,MS2       ; MESSAGE 'TX=1 RX=0'
        .SYSTM          ; CALL RDOS
        .WRL 1          ; WRITE IT
        JMP @.ERROR     ; OK
        .SYSTM          ; GET REPLY
        .GCHAR          ; OK
        JMP @.ERROR     ; ERROR
        LDA 1,ONE       ; ONE ?
        SUBZ# 1,0,SNR   ; ?
```

```
        JMP TXMODE      ; TRANSMIT MODE
        LDA 1,ZERO      ; Ø ?
        SUBZ# 1,Ø,SZR   ; ?
        JMP MESS1       ; MESSAGE
RXMODE: JSR @.SETUP     ; CALL SETUP
        LDA Ø,@.ERR1    ; SETUP ERROR
        MOVZ Ø,Ø,SZR    ; NO ERROR ?
        JMP @.ERROR     ; ERROR
        LDA Ø,CNUM      ; COUNT NUMBER
        STA Ø,COUNT     ; STORE IT
BAK1:   JSR @.RXMT      ; CALL RECEIVE
        LDA Ø,@.NEW     ; GET REPLY
        LDA 1,A4        ; ENQ ?
        SUBZ# Ø,1,SNR   ; WELL ?
        JMP PØ          ; NO !
        ISZ COUNT       ; DONE YET ?
        JMP BAK2        ; NO SO DO AGAIN
        LDA Ø,MS3       ; FINISH MESSAGE
        .SYSTM          ; RDOS
        .WRL 1          ; TELL EM
        JMP @.ERROR     ; ZAP IT
        .SYSTM
        .RTN
        .RTN            ; DIE
        ;
        ;
BAK2:   JSR @.TENQ      ; SEND ENQUIRE
        JMP BAK1        ; GO BACK
        ;
        ;
PØ:     JSR @.TACK      ; SEND ACK
        JSR @.RXMT      ; GET REPLY
        LDA Ø,@.NEW     ; GOT IT, NOW WHAT ?
        LDA 1,A4        ; ENQ ?
        SUBZ# 1,Ø,SZR   ; WELL ?
        JMP P1          ; OVER
        LDA 1,LAST      ; WHAT WAS THE LAST ONE ?
        MOV 1,1,SZR     ; Ø=NAK 1=ACK
        JMP J1
        JSR @.TNAK      ; NEGATIVE
        JMP PØ+1
J1:     JSR @.TACK      ; POSITIVE
        JMP PØ+1                ; AND BACK AGAIN
        ;
        ;
P1:     LDA 1,A1        ; HMMM
        SUBZ# 1,Ø,SZR   ; BLOCK IN ERROR ?
        JMP P2          ; NOT YET ANYWAY
        JSR @.TNAK      ; NEGATIVE ACK
        SUBZ Ø,Ø        ; Ø
        STA Ø,LAST      ; LAST=NAK
        JMP PØ+1                ; BACK AGAIN
        ;
        ;
P2:     MOVZ Ø,Ø,SZR    ; GOOD BLOCK ?
        JMP P3          ; NEGATIVE
        LDA 1,@.TEMP8   ; NO OF BYTES
        LDA Ø,@.RARY    ; RECEIVE ARRAY
        MOVZL Ø,Ø       ; BYTE POINTER
        .SYSTM          ; GET THE DATA ON DISK
        .WRS 4          ; DONE
        JMP @.ERROR     ; ERROR ?
        SUBZL Ø,Ø       ; +1
        STA Ø,LAST      ; LAST = O.K.
        JMP PØ          ; BACK FOR MORE
        ;
```

```
                ;
P3:             LDA 1,A5        ; EOB
                SUBZ# 1,0,SZR   ; ?
                JMP P4          ; DONT RECOGNISE IT
                JSR @.TACK      ; CONFIRM IT
                LDA 0,@.MS3     ; DONE
                .SYSTM          ; RDOS
                .WRL 1          ; O.K.
                JMP @.ERROR     ; .
                .SYSTM
                .RTN
                .RTN            ; DEAD
                ;
                ;
P4:             JSR @.TENQ      ; SEND ENQ BLOCK
                JMP P0+1                ; MORE
TXMODE:         JSR @.SETUP     ; CALL SETUP
                LDA 0,@.ERR1    ; SETUP ERROR
                MOVZ 0,0,SZR    ; NO ERROR ?
                JMP @.ERROR     ; ERROR
                LDA 0,CNUM      ; ERROR COUNT
                STA 0,COUNT     ; STORE IT
TXAG:           JSR @.TENQ      ; ENQUIRY BLOCK
                JSR @.RXMT      ; RECEIVER
                LDA 0,@.NEW     ; GET RECEIVER STATUS
                LDA 1,A2        ; STATUS 2
                SUBZ# 1,0,SNR   ; IS IT
                JMP NEXTB       ; CONTINUE
                ISZ COUNT       ; COUNT=0 ?
                JMP TXAG        ; NO SO AGAIN
                LDA 0,@.MS3     ; MESSAGE
                .SYSTM          ; CALL RDOS
                .WRL 1          ; SENT IT
                JMP @.ERROR     ; DONE
                .SYSTM
                .RTN
                .RTN            ; DEAD NOW
                ;
                ; END O' THE LINE
                ;
NEXTB:          JSR @.READL     ; GET A LINE
                LDA 0,@.ECODE   ; GET CODE
                LDA 1,A6        ; 6
                SUBZ# 1,0,SZR   ; WELL ?
                JMP J2          ; APPARENTLY NOT
                ISZ FIN         ; SIGNAL DONE
                JMP NEXTA       ; CONTINUE
J2:             MOVZ 0,0,SZR    ; 0 PERHAPS ?
                JMP @.ERROR     ; FUNNY ERROR !
NEXTA:          JSR @.TDUM
                JSR @.TDUM
                JSR @.TDUM
```

```
          JSR @.TDUM
          JSR @.TDUM        ; WIERD MAN !
          SUBZL 0,0         ; +1
          STA 0,@.CONTR     ; DATA BLOCK
          JSR @.TXMT        ; GO.. GO... GO .....
NEXTC:    JSR @.RXMT        ; GET THE REPLY
          LDA 0,@.NEW       ; OK SO THATS THE REPLY
          LDA 1,A2          ; 2 ?
          SUBZ# 0,1,SZR     ; WELL ?
          JMP 01            ; NEGARORY RUBBER DUCK
          LDA 1,FIN         ; END OF BLOCK ?
          MOV 1,1,SNR       ; END ?
          JMP NEXTB         ; GET THE NEXT BLOCK
          JMP THEEND        ; ZAP.A.ROONY
          ;
          ;
01:       LDA 1,A3          ; NAK (TOO YOU TOO)
          SUBZ# 1,0,SZR     ; WELL ?
          JMP 02            ; NO !
          JMP NEXTA         ; TX AGAIN O.K.
          ;
          ;
02:       LDA 1,A4          ; ENQ
          SUBZ# 1,0,SZR     ; WELL ?
          JMP 03            ; NO !
          JMP NEXTA         ; TX AGAIN AND AGAIN AND AGAIN AND AGAIN
          ;
          ;
03:       JSR @.TENQ        ; SUMMUT UP 'ERE
          JMP NEXTC         ; WAIT FOR REPLY
          ;
          ;
THEEND:   JSR @.TEOB        ; END OF BLOCK MUN
          JSR @.RXMT        ; GET REPLY
          LDA 0,@.MS4       ; GET END MESSAGE
          .SYSTM            ; SEND IT
          .WRL 1            ; REPLY
          JMP @.ERROR       ; ERROR
          .SYSTM            ; IGNORE IT ANYWAY O.K.
          .RTN              ; BYE BYE
          .RTN              ; BAG SOME Z'S THERE BOY
          ;
          ;
ERROR:    .SYSTM
          .ERTN             ; KILL IT
          ;
          ;
          .ZREL
          ;
```

```
                ;
.MS3:     MS3
.MS4:     MS4
.ERROR:   ERROR
.TEMP8:   TEMP8
.DARAY:   DARAY
.TNAK:    TNAK
.TACK:    TACK
.TENQ:    TENQ
.TARY:    TARY
.RARY:    RARY
.SETUP:   SETUP
.TXMT:    TXMT
.RXMT:    RXMT
.READL:   READL
.TDUM:    TDUM
.TEOB:    TEOB
.ERR1:    ERR1
.NEW:     STAT1
.ECODE:   ECODE
.CONTR:   CONTR
ONE:      61
ZERO:     60
CNUM:     177772
COUNT:    0
FIN:      0
LAST:     0
A1:       1
A2:       2
A3:       3
A4:       4
A5:       5
A6:       6
          .END START
```

```
x         .TITLE TDUM
          .EXTD TXMT
          .ENT TDUM
          .EXTN TARY CONTR
          .TXTM 1
          .NREL
TDUM:     STA 3,SAV3
          LDA 0,WORK        ; BLANK ARRAY
          LDA 2,.TARY       ; ARRAY ADDRESS ADDRESS
          LDA 3,@.TARY      ; ACTUAL ADDRESS
          STA 3,TEMP        ; STORE IT
          STA 0,0,2         ; ALTER ADDRESS TO WORK AREA
          SUB 0,0           ; CONTROL BLOCK
          STA 0,@.CONTR     ; DONE
          JSR @.TXMT        ; TRANSMIT
          LDA 3,TEMP        ; GET ADDX BACK
          LDA 2,.TARY       ; ADDRESS
          STA 3,0,2         ; PUT IT BACK
          LDA 3,SAV3        ; RETURN
          JMP 0,3           ; BYE
          ;
          ;
SAV3:     0
TEMP:     0
.TARY:    TARY
.TXMT:    TXMT
.CONTR:   CONTR
WORK:     .+1
          .BLK 50
          .END
```

```
Lx          .TITLE FISERV
            .NREL
            .EXTN .UIEX
            .ENT PART
            .ENT AIN1
            .ENT TFLG RFLG MFLG TDAT RDAT STAT
            .DUSR ULM=34
AIN1:       STA 3,SAV3
            STA 2,SAV2
            DIA 0,ULM           ; GET LINE/SECTION
            MOVZR 0,0,SZC       ; TX/RX ?
            JMP TSERV           ; TX
            DIC 0,ULM           ; RX/MODEM ?
            MOVZR 0,0,SZC       ; RX
            JMP MSERV           ; MODEM
            STA 0,STAT          ; SAVE STATUS
            DIBC 0,ULM          ; GET DATA
            STA 0,RDAT          ; STORE IT
            LDA 3,SYN           ; SYNC
            SUBZ# 3,0,SZR       ; 0 ?
            JSR CRC             ; CALC CRC
            SUBZL 0,0           ; INTERRUPT=1
            STA 0,RFLG          ; OK
            JMP FINISH          ; END
            ;
            ;
MSERV:      STA 0,STAT          ; SAVE STATUS
            SUBZL 0,0           ; +1
            STA 0,MFLG          ; INTERRUPT=1
            NIOC ULM            ; CLEAR INTERRUPT
            JMP FINISH          ; END
            ;
            ;
TSERV:      NIOC ULM            ; CLEAR INTERRUPT
            LDA 0,TDAT          ; GET TX DATA
            DOB 0,ULM           ; TRANSMIT IT
            LDA 3,SYN           ; SYNC
            SUBZ# 3,0,SZR       ; ?
            JSR CRC             ; CALC CRC
            SUBZL 0,0           ; +1
            STA 0,TFLG          ; INTERRUPT=1
            ;
            ;
FINISH:     LDA 2,SAV2          ; GET 2
            LDA 3,SAV3          ; AND 3
            SUB 1,1             ; FORCE NO RE-SCHEDULING
            .UIEX               ; EXIT
            ;
            ;
CRC:        LDA 0,PART          ; GET PARTIAL
            DOBS 0,35           ; START
            SKPBZ 35            ; BUSY ?
            JMP .-1             ; WAIT
            DIBC 0,35           ; GET NEW ONE
            STA 0,PART          ; STORE IT
            JMP 0,3             ; RETURN
            ;
            ; DATA
            ;
SAV2:       0
SAV3:       0
RFLG:       0
MFLG:       0
TFLG:       0
TDAT:       0
RDAT:       0
STAT:       0
            ;
PART:       0
SYN:        26
            .END
```

```
          .TITLE FRXMT
          .ENT RXMT
          .EXTN RFLG TFLG MFLG TDAT RDAT
          .ENT TEMP0
          .ENT STAT1 WORDS
          .EXTN PART
          .EXTN RARY
          .TXTM 1
          .DUSR ULM=34
          .NREL
SAV3:     0
.PART:    PART
.RARY:    RARY
          ;
          ; RECEIVE PROGRAM ERROR RETURNS ARE :-
          ;
          ; 0=DATA BLOCK WITH NO ERRORS
          ; 1=DATA BLOCK WITH BURST ERROR
          ; 2=CONTROL BLOCK ACK
          ; 3=CONTROL BLOCK NAK
          ; 4=CONTROL BLOCK ENQ
          ; 5=SYNC CHARACTERS
          ; 6=UNDEFINED CONTROL SEQUENCE
          ; 7=TIMEOUT
          ;
          ;
RXMT:     STA 3,SAV3        ; SAVE ACC 3
          NIOC 35           ; CLEAR CRC
          LDA 0,RTSOFF      ; RTS='0'
          DOB 0,ULM         ; SET IT AND ULM=ON
          LDA 0,SL          ; SYNC LINE
          DOA 0,ULM         ; SET IT
          SUBZL 0,0         ; +1
          DOC 0,ULM         ; LINE IS ON
          SUBZ 1,1          ; GENERATE +0
          STA 1,STAT1       ; STATUS
          STA 1,START       ; START=0
          STA 1,ERROR       ; ERROR=0
          STA 1,END         ; END  =0
          STA 1,LASTD       ; LASTD=0
          STA 1,@.PART      ; CLEAR CRC PARTIAL
          STA 1,TEMP0       ; COUNTER=0
          LDA 2,@.RARY      ; GET ADDRESS
          ;
NEXT:     JSR RCHAR         ; GET A CHAR
          MOVS 0,0          ; TOP BYTE
          STA 0,0,2         ; STORE IT
          JSR RCHAR         ; GET A CHAR
          LDA 1,0,2         ; GET LAST VALUE
          ADDZ 1,0          ; ADD IN NEW
          STA 0,0,2         ; STORE IT
          INC 2,2           ; BUMP POINTER
          LDA 1,TEMP0       ; CURRENT NO.
          LDA 0,MAX         ; MAX ALLOWED
          SUBZL# 0,1,SNC    ; FULL ?
          JMP FIN2
          LDA 1,END         ; DONE ?
          MOVZ 1,1,SNR      ; WELL ?
          JMP NEXT          ; LOOP
          ;
          ; CRC NOW
```

```
                ;
                LDA 0,@.PART      ; GET PARTIAL
                STA 0,TPART       ; STORE FOR LATER
                JSR RCHAR         ; GET NEXT BYTE
                MOVS 0,0          ; HI BYTE
                STA 0,0,2         ; STORE IT
                JSR RCHAR         ; GET NEXT ONE
                LDA 1,0,2         ; GET LAST ONE
                ADDZ 1,0          ; ADD IN NEW
                STA 0,0,2         ; STORE WHOLE ONE
                ;
                ; ENDIT ALL
                ;
FINISH:         LDA 0,SL          ; SYNC LINE
                DOA 0,ULM         ; SET IT
                SUBZ 0,0          ; +0
                DOC 0,ULM         ; SECTION = OFF
                ;
                ;
                SUBZL 1,1         ; +1
                LDA 0,TEMP0       ; NO OF WORDS GOT
                SUBZ 1,0          ; NO.-1
                SUBZ 1,0          ; NO. OF WORDS -1 (BYTES-2)
                LDA 3,STAT1       ; GET STATUS
                MOV 3,3,SZR       ; 0 ?
                JMP FIN1          ; END IT
                STA 0,TEMP8       ; STORE IT
                LDA 0,TPART       ; GET CRC
                LDA 1,0,2         ; GET CRC
                SUBZ 3,3          ; +0
                SUBZ# 1,0,SZR     ; SAME ?
                INC 3,3           ; +1
                STA 3,STAT1       ; STORE IT
                ;
FIN1:           LDA 1,STAT1       ; GET STATUS
                LDA 3,SAV3        ; GET ACC 3
                LDA 1,TEMP8       ; NO. OF BYTES
                MOVZR 1,1         ; NO. OF WORDS
                STA 1,WORDS       ; STORE IT
                JMP 0,3           ; RETURN
                ;
                ; BLOCK TOO LONG
                ;
FIN2:           LDA 0,A1          ; ERROR
                STA 0,STAT1       ; DONE
                JMP FINISH        ; END
                ;
                ; SOME DATA
                ;
WORDS:          0
TEMP0:          0
START:          0
ERROR:          0
STAT1:          0
END:            0
MAX:            500.
LASTD:          0
SL:             20
RTSOFF:         100001
TPART:          0
TEMP8:          0
                ;
                ; BASIC RX ROUTINE
                ;
```

```
        ;
RCHAR:  STA 3,SAV33
        SUBZ 3,3            ; GENERATE +0
        STA 3,TEMP          ; TIMER
        LDA 3,DEL1          ; DELAY=200
        STA 3,TEMP1         ; TIMER
        JSR TIME1           ; TIMEOUT
        SUBZ 3,3            ; ZERO FLAG
        STA 3,@.RFLG        ; CLEAR FLAG
        LDA 0,@.RDAT        ; GET DATA
        LDA 3,DLE           ; DLE
        SUBZ# 3,0,SZR       ; IS IT ?
        JMP NODLE           ; NO !
        LDA 3,LASTD         ; LAST ONE A DLE ?
        MOV 3,3,SZR         ; WELL ?
        JMP ONDLE           ; YES !
        ISZ LASTD           ; NO SO SET IT
        JMP EXIT1           ; LEAVE
        ;
        ;
NODLE:  LDA 3,LASTD         ; WAS LAST ONE A DLE
        MOVZ 3,3,SNR        ; WELL ?
        JMP EXIT1           ; WAIT FOR IT !
        SUBZ 3,3            ; +0 .
        STA 3,LASTD         ; CLEAR DLE FLAG
        LDA 1,STX           ; START OF TEXT ?
        SUBZ# 1,0,SNR       ; ?
        JMP STX1            ; YES !
        LDA 1,ETX           ; END OF TEXT ?
        SUBZ# 1,0,SNR       ; ?
        JMP ETX1            ; YES
        LDA 1,SYN           ; SYNC
        SUBZ# 1,0,SNR       ; OK
        JMP SYN1            ; ZAP
        LDA 1,ACK           ; ACK
        SUBZ# 1,0,SNR       ; WELL
        JMP ACK1                    ; ACK
        LDA 1,NAK           ; NAK ?
        SUBZ# 1,0,SNR       ; WELL
        JMP NAK1            ; OK
        LDA 1,ENQ           ; ENQ?
        SUBZ# 1,0,SNR       ; WELL
        JMP ENQ1            ; YES
        ;
        ; ERROR UNDEFINED CONTROL CHARS
        ;
        LDA 1,A6            ; ERROR CODE 6
        STA 1,STAT1         ; STORE IT
        JMP EXIT            ; GONE
        ;
        ; CONTROL CHARACTER SECTION
        ;
STX1:   ISZ START           ; START COLLECTING CHARACTERS
        MOV 0,0
        SUBZ 0,0            ; +0
        STA 0,@.PART        ; CLEAR CRC
        JMP RCHAR+1         ; GET ANOTHER
        ;
ETX1:   ISZ END             ; FINISH
        MOV 0,0
        JMP EXIT            ; OK
        ;
SYN1:   LDA 1,A5            ; ERROR CODE 5
        STA 1,STAT1         ; O.K.
        JMP EXIT            ; LEAVE
```

```
        ;
ACK1:   LDA 1,A2        ; ERROR CODE 2
        STA 1,STAT1     ; O.K.
        JMP EXIT        ; LEAVE
        ;
NAK1:   LDA 1,A3        ; ERROR CODE 3
        STA 1,STAT1     ; O.K.
        JMP EXIT        ; LEAVE
        ;
ENQ1:   LDA 1,A4        ; ERROR CODE 4
        STA 1,STAT1     ; O.K.
        JMP EXIT        ; LEAVE
        ;
EXIT:   LDA 3,SAV33     ; GET 3 BAK
        ISZ TEMP0       ; COUNTER
        JMP 0,3         ; LEAVE
        ;
EXIT1:  LDA 1,START     ; HAVE WE STARTED YET ?
        MOVZ 1,1,SZR    ; WELL ?
        JMP EXIT        ; YES SO LEAVE
        JMP RCHAR+1     ; NO SO GET ANOTHER
        ;
ONDLE:  SUBZ 3,3        ; CLEAR DLE FLAG
        STA 3,LASTD     ; DONE
        JMP RCHAR+1     ; GET ANOTHER CHARACTER
        ;
        ; TEST AND TIME ROUTINE
        ;
TIME1:  LDA 1,@.RFLG    ; GET THE FLAG
        MOV 1,1,SZR     ; IS IT 0
        JMP 0,3         ; NO SO NORMAL RETURN
        DSZ TEMP        ; TIMER
        JMP TIME1       ; O.K.
        DSZ TEMP1       ; TIMER
        JMP TIME1       ; O.K.
        LDA 1,A7        ; ERROR CODE 7
        STA 1,STAT1     ; STATUS=TIMEOUT
        JMP FINISH      ; EXIT
        ;
        ; DATA
        ;
STX:    2
ETX:    3
SYN:    26
ACK:    6
NAK:    25
ENQ:    5
DLE:    20
.RFLG:  RFLG
.RDAT:  RDAT
SAV33:  0
TEMP:   0
TEMP1:  0
FIVE:   5
A1:     1
A2:     2
A3:     3
A4:     4
A5:     5
A6:     6
A7:     7
DEL1:   60
        .END
```

```
          .TITLE FSETUP
          .NREL
          .TXTM 1
          .ENT SETUP IN1
          .ENT ERR1
          .EXTN AIN1
          .DUSR ULM=34
SETUP:    STA 3,SAV3              ; SAVE 3
          SUB 2,2                 ; CLEAR ERROR
          STA 2,ERR1              ; =0
          LDA 0,UDEV              ; DEVICE 34
          LDA 1,IN1               ; DCI
          .SYSTM            ; CALL RDOS
          .IDEF             ; CALL IDEF
          JMP ERROR               ; ERROR ?
NRET:     NIOS ULM                ; SETUP
          LDA 0,SL                ; LINE NO.
          LDA 1,LC                ; CHARACTERISTICS
          LDA 2,SYN               ; SYNC CHAR
          LDA 3,MCS               ; MODEM CONTROL
          SKPBZ ULM               ; WAIT
          JMP .-1                 ; OK
          DOA 0,ULM
          DOC 1,ULM
          DOC 2,ULM
          DOB 3,ULM               ; DONE
          LDA 1,DLE               ; DATA LINK
          DOC 1,ULM               ; SETIT
          INC 0,0                 ; TX SECTION
          DOA 0,ULM               ; SET SECTION
          DOC 2,ULM               ; SET SYNC
          DOC 1,ULM               ; SET DLE
          NIOC ULM                ; ONLINE
          LDA 3,SAV3              ; GET 3
          JMP 0,3
          ;
          ; ERROR ROUTINE
          ;
ERROR:    LDA 3,SAV3
          STA 2,ERR1              ; =0
          JMP NRET
          ;
          ; DATA
          ;
SAV3:     0
MASK:     200
SL:       20
LC:       101030
SYN:      40026
MCS:      100001
DLE:      140020
ERR1:     0
UDEV:     34
          ;
          ; INTERRUPT DEVICE TABLE FOR RDOS
          ;
IN1:      .+1
          0
          200
          AIN1
          ;
          ; END
          ;
          .END
```

Presnell, R.I. et al. (1959), VHF and UHF radar observations of the aurora at college, Alaska. J. Geophys. Res., 64, 1179.

Rather, E. (1977), FORTH a fresh approach to programming. FORTH. Inc. publication.

Rogister, A. et al. (1970), Type II irregularities in the equatorial electrojet.